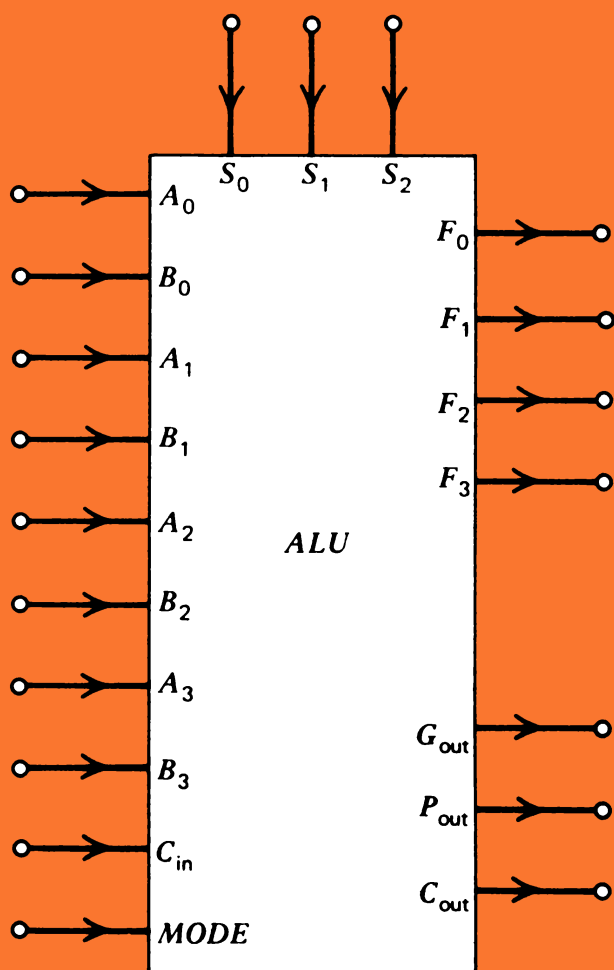
A. BARNA - D. I. PORAT

MICROCOMPUTER E MICROPROCESSORI

Traduzione a cura dell' ing. Franco Govoni

EDIZIONI C.E.L.I.

MICROCOMPUTER E MICROPROCESSORI

Arpad BARNA - Dan I. PORAT

MICROCOMPUTER E MICROPROCESSORI

Edizione italiana a cura
dell'ing. FRANCO GOVONI

EDIZIONI C.E.L.I. BOLOGNA

PREFAZIONE

L'introduzione di un numero sempre crescente di microcomputer e di microprocessori ha condotto ad una grande varietà delle loro applicazioni. Il progetto di sistemi basati su microprocessori richiede però la conoscenza di diverse discipline, fra queste il progetto logico, i sistemi digitali, l'architettura dei computer, le tecniche di programmazione, e in minor grado il progetto dei circuiti elettronici e la tecnologia dei semiconduttori. Questo libro introduttivo è scritto per chi non ha conoscenze sufficienti in tutte queste aree e desidera invece apprendere le tecniche richieste pr l'uso efficiente dei microcomputer e dei microprocessori.

La materia è presentata in tre fasi. I primi tre capitoli forniscono una descrizione sommaria della hardware e del software di base, i successivi cinque capitoli forniscono invece dettagli del funzionamento, mentre l'ultimo capitolo presenta alcuni complementari di software. Ogni capitolo può stare a sè, con un minimo di richiami ad altre parti del libro, così che il lettore può omettere interi capitoli senza pregiudicare la leggibilità del libro e può facilmente utilizzare il materiale per la consultazione.

I 120 fra esempi e problemi inclusi nel testo rendono il libro particolarmente adatto per lo studio autonomo, fornendo una solida base per

comprendere le caratteristiche di un grande numero di microcomputer e di microprocessori oggi disponibili. La bibliografia alla fine del libro indica ulteriore materiale relativo agli argomenti discussi. Sono anche riportate le risposte ad alcuni problemi prescelti.

ARPAD BARNA
DAN I. PORAT

Stanford, California
Settembre 1975

INDICE GENERALE

LISTA DELLE ABBREVIAZIONI

ALU	*arithmetic-logic unit*
ASCII	*American Standard Code for Information Interchange*
BCD	*binary-coded decimal*
CCD	*charge-coupled device*
CPU	*central processor unit*
DMA	*direct memory access*
EPROM	*erasable programmable read-only memory*
FIFO	*first-in-first-out*
I/O	*input/output*
LIFO	*last-in-first-out*
lsb	*least significant bit*
MAR	*memory address register*
MDR	*memory data register*
MPX	*multiplexer*
MUX	*multiplexer*
msb	*most significant bit*
PLA	*programmable logic array*
PROM	*programmable read-only memory*
RAM	*random-access memory*
ROM	*read-only memory*

1. INTRODUZIONE

Una delle tappe fondamentali lungo la via dello sviluppo dei sistemi a funzionamento automatico è stata l'introduzione del *computer*. A differenza degli altri sistemi, in un computer la sequenza delle operazioni è controllata da un *programma memorizzato* all'interno.

> **Esempio 1.1.** Il traffico dei veicoli all'incrocio di una strada principale e di una secondaria è regolato da un semaforo comandato da un *controller* che ha un ciclo di funzionamento di 60 secondi. Le luci per la strada principale restano verdi per 30 secondi, e poi gialle per 5 secondi, rosse per 20 secondi e infine gialle per 5 secondi. Per quanto semplice, il *controller* può essere considerato un computer a programma memorizzato.

Nell'accezione corrente del termine, però, un computer a programma memorizzato possiede una caratteristica addizionale: è capace di *diramazioni* fra vari segmenti del programma. Tali diramazioni, o *decisioni*, possono essere controllate dal risultato di calcoli precedenti, o anche dalle informazioni ricevute da un *dispositivo di ingresso* del computer.

> **Esempio 1.2.** Il *controller* dell'esempio 1.1 viene ampliato aggiungendo due sensori di veicoli collegati con la funzione di dispositivi di ingresso. Questi sensori, posti nella strada secondaria, segnalano la presenza di veicoli in attesa del cambio della luce del semaforo. Alla fine dei 30 secondi di luce verde per la strada principale, il *controller* interroga i sensori e cambia le luci solo se ci sono veicoli in attesa su una strada secondaria.

I computer digitali a programma memorizzato si sono largamente diffusi durante le ultime due decadi. Questo fatto è stato determinato principalmente da sviluppi tecnologici come l'introduzione dei transistori che ora sono presenti in ogni parte dei computer, dal miglioramento degli elementi usati nella *memoria*, dall'aumentata affidabilità dei *dispositivi periferici* elettromeccanici, e infine dall'uso crescente di circuiti integrati. Attualmente i computer digitali comprendono computer per impieghi speciali (*special-purpose*), costruiti per un solo tipo di impiego, e computer per impieghi generali (*general-purpose*), utilizzati in molte aree diverse, come il controllo, l'elaborazione dei dati (*data processing*), e il calcolo scientifico.

Parallelamente alla crescita di affidabilità, potenza e facilità di impiego dei computer per impieghi generali, si andava sviluppando il *mini-computer* per impieghi generali, meno potente, ma più piccolo e meno costoso. Principalmente per il loro costo minore, i mini-computer sono entrati in numerose applicazioni che in precedenza erano dominio esclusivo di piccoli computer per impieghi speciali. La lacuna residua che separava i computer per impieghi generali da quelli per impieghi speciali e dai *controller* sta ora per essere colmata ad opera dei *microcomputer*, i più recenti e i più piccoli computer per impieghi generali.

I primi microcomputer erano soltanto dei calcolatori aritmetici; ora invece i microcomputer stanno occupando ed allargando il ruolo di molti minicomputer e dei computer per impieghi speciali, in particolare quello dei *controller a logica cablata* per impieghi speciali.

> **Esempio 1.3.** Sul percorso di una metropolitana sono installati dispositivi di sicurezza ausiliari sincronizzati. Su ogni tratto del binario è installato un tale dispositivo, che segnala i treni che entrano e lasciano il tratto. Nella realizzazione iniziale ciascun dispositivo di sicurezza usava un controller a logica cablata specializzato. In conseguenza dei numerosi casi speciali derivanti dalla diversità delle diramazioni del binario, i controller però non potevano essere tutti uguali. Nella realizzazione finale i controller a logica cablata furono quindi sostituiti da microcomputer e i casi speciali furono affrontati mediante differente programmazione.

La semplicità e il costo ridotto che invitano ad una larga applicazione dei microcomputer hanno però come contropartita tecniche di programmazione più difficili e primitive di quelle richieste dai mini-

computer. Inoltre, in un microcomputer i problemi relativi alla circuiteria, o *hardware*, spesso si intersecano con quelli relativi alla programmazione, o *software*, molto più che in un minicomputer. Di conseguenza, il lavoro di sviluppo di un sistema che impiega un microcomputer, sebbene talvolta possa essere ripartito fra esperti di hardware e esperti di software, quasi sempre richiede la conoscenza di base di entrambi gli aspetti. Per questa ragione, nella maggior parte di questo libro i due aspetti sono toccati insieme, con l'intento di fornire un'introduzione equilibrata al hardware e al software relativi alle applicazioni dei microcomputer.

2. STRUTTURA DI BASE DEI MICROCOMPUTER E DEI MICROPROCESSORI

In fig. 2.1. è mostrato il diagramma a blocchi semplificato di un microcomputer. Esso è limitato a tre blocchi funzionali: la *sezione di ingresso-uscita*, l'*unità centrale* e la *memoria centrale.* *
Uno schema a blochi più dettagliato è riportato nel Cap. 8.

FIG. 2.1. - Schema a bocchi semplificato di un microcomputer. Le linee di interconnessione possono rappresentare collegamenti multipli.

2.1. Sezione di ingresso-uscita

La *sezione di ingresso-uscita* (*I/O*: *input-output*) connette il microcomputer con i dispositivi di ingresso e d'uscita, detti anche dispositivi periferici.

> **Esempio 2.1.** Un calcolatore tascabile ha 10 tasti numerici contrassegnati con le cifre da 0 a 9, cinque tasti di funzioni

* Uno schema a blocchi più dettagliato è riportato nel Cap. 8.

+, −, ×, ÷, e =, un *visualizzatore* (*display*) con sei cifre decimali, e incorpora un microcomputer che opera su i dati e li memorizza. I tasti costituiscono i dispositivi di ingresso e il display costituisce il dispositivo di uscita.

Nella fig. 2.2 è mostrato il diagramma a blocchi semplificato di una sezione I/O. La selezione dei dispositivi I/O è effettuata dai *multi-*

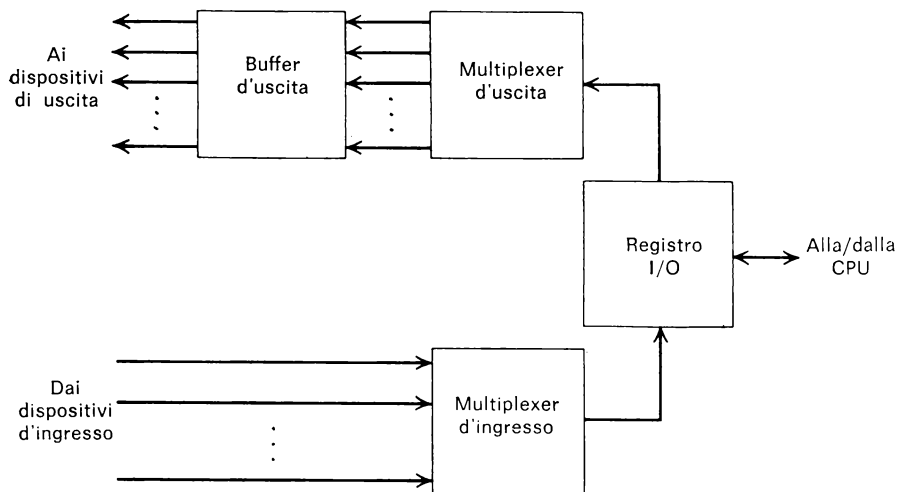



FIG. 2.2. - Schema a blocchi semplificato di una sezione I/O

plexer (abbreviati in MPX o MUX) di ingresso e di uscita, detti anche *selettori dei dati.* L'informazione in uscita è memorizzata nei *buffer d'uscita.* Il *registro I/O* fornisce una memorizzazione temporanea durante lo scambio di informazioni fra la CPU e la sezione I/O.

Esempio 2.2. Un controller del traffico ad un incrocio stradale impiega un microcomputer al quale sono connessi come dispositivi di ingresso quattro sensori che segnalano la presenza dei veicoli. I dispositivi di uscita sono costituiti da quattro semafori, che il microcomputer deve controllare costantemente.

Poiché la velocità dei veicoli è limitata, ogni sensore segnala la presenza di un veicolo per almeno 0,1 secondo. Di

conseguenza, i quattro sensori possono essere scanditi in sequenza dal microcomputer, se ogni sensore è interrogato ad un ritmo uniforme di almeno 10 letture al secondo. La sezione I/O del microcomputer che agisce da controller del traffico può essere rappresentata come in fig. 2.2.

Le caratteristiche e le connessioni dei dispositivi di ingresso e d'uscita sono discusse nel Cap. 4.

2.2. Unità centrale

La struttura interna di una *unità centrale* (*CPU: central processor unit*) varia largamente da un microcomputer ad un altro. La fig. 2.3 mostra lo schema a blocchi di una semplice CPU, costituita da

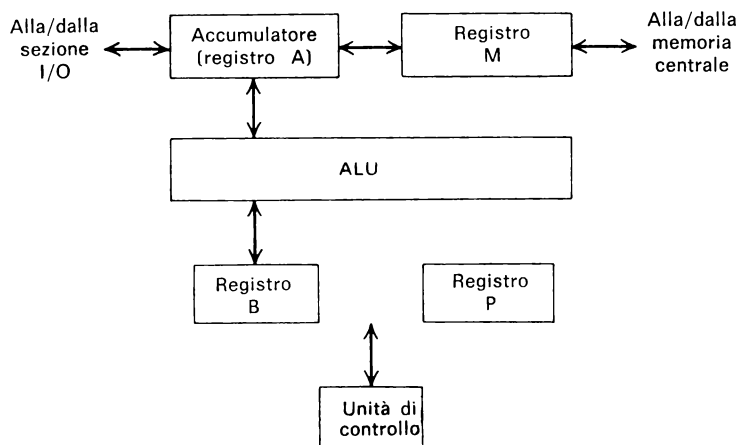



FIG. 2.3. - Schema a blocchi semplificato di una CPU. Non sono indicati i collegamenti dell'unità di controllo e del registro P.

un'*unità aritmetica-logica* (*ALU: arithmetic-logic unit*), da alcuni *registri* e da un'*unità di controllo*. Il numero delle linee che collegano l'ALU, l'*accumulatore* (registro A) e i registri B e M è stabilito dalla « *lunghezza della parola* », che è il numero massimo di cifre binarie (bit) che l'unità aritmetico-logica può trattare in parallelo.

Unità aritmetico-logica

L'ALU opera su uno o su due numeri; può eseguire operazioni aritmetiche come l'addizione e la sottrazione, e operazioni logiche come il confronto. La struttura e il funzionamento di una ALU sono discusse nel Cap. 6.

Registri

La CPU comprende diversi registri, spesso indicati come *registri dei dati* (*data register*), *registri operativi* o *registri di lavoro* (*working register*), o come *memoria scratch-pad.* Il registro A (accumulatore) e il registro B costituiscono la memoria per i dati su cui agisce la ALU.

> **Esempio 2.3.** Nella CPU di fig. 2.3, l'addizione è eseguita sommando il contenuto del registro B al contenuto dell'accumulatore A e collocando il risultato dell'addizione nell'accumulatore A.

La capacità di bit dell'accumulatore e quella del registro B sono stabilite dalla lunghezza della parola. In alcune operazioni aritmetiche i due registri possono essere collegati insieme e formare un unico registro di capacità doppia.

> **Esempio 2.4.** In una CPU con parole lunghe 16 bit la moltiplicazione viene eseguita moltiplicando il contenuto dell'accumulatore per il contenuto del registro B. Il risultato della moltiplicazione è un numero di 32 bit che viene memorizzato collocando i 16 bit più significativi nell'accumulatore e i 16 bit meno significativi nel registro B.

La CPU di fig. 2.3 comunica con i dispositivi I/O per mezzo della sezione I/O e comunica con la memoria centrale attraverso il registro M.

> **Esempio 2.5.** Un microcomputer che incorpora la CPU di fig. 2.3 è usato in un controller industriale di temperatura. Il controller misura la temperatura per mezzo di cinque sen-

sori, ciascuno dei quali è interrogato ad un ritmo uniforme di 12 letture al minuto. La temperatura è controllata per mezzo di un riscaldatore elettrico basato sulle ultime tre letture di tutti e cinque i sensori di temperatura.

I dati provenienti dai sensori sono trasmessi alla memoria principale attraverso la sezione I/O, l'accumulatore e il registro M. I dati di temperatura relativi alle ultime tre letture vengono elaborati dall'ALU, il che richiede un ulteriore trasferimento di dati fra la memoria principale e la CPU attraverso il registro M. Il risultato dell'elaborazione, che costituisce l'informazione per il controllo, è inviata al riscaldatore attraverso la sezione I/O.

Il registro P è il *contatore di programma* (*program counter*) e stabilisce la successione delle operazioni del microcomputer. Il contatore di programma conta in salita per uno se non è comandato altrimenti. Ciascun passo del contatore può corrispondere ad una sola operazione, come l'addizione di due numeri, o può corrispondere ad una intera sequenza di operazioni.

Esempio 2.6. Un semplice sistema di riscaldamento è costituito da un termostato, da un riscaldatore che alternativamente viene inserito e disinserito, e dalla CPU di fig. 2.3. Il valore di temperatura desiderato è trasferito dalla memoria nel registro M. Il ciclo di controllo è sequenzializzato dal contatore del programma, ossia dal registro P. Una sequenza di controllo semplificata è delineata nella Tav. 2.1; essa viene iniziata ogni 10 secondi ponendo a zero il contenuto del registro P.

Unità di controllo

La funzione principale dell'*unità di controllo* (*control unit*) in un microcomputer è di fornire l'opportuna direzione a tutto il funzionamento del computer.

Esempio 2.7. L'addizione di due numeri nella CPU di fig. 2.3 è eseguita come descritto nell'Esempio 2.3. L'unità di controllo dapprima costituisce le *vie dei dati* (*data paths*) per

TABELLA 2.1

Sequenza di controllo semplificata per il sistema di riscaldamento dell'esempio 2.6

CONTENUTO REGISTRO P	OPERAZIONE
0	Start
1	Trasferisci la temperatura desiderata dal registro M al registro B
2	Leggi il termostato e trasferisci la lettura nell'accumulatore attraverso la sezione I/O
3	Confronta nella ALU il contenuto dell'accumulatore con il contenuto del registro B
4	Inserisci il riscaldatore attraverso la sezione I/O se il contenuto dell'accumulatore è minore del contenuto del registro B
5	Disinserisci il riscaldatore attraverso la sezione I/O se il contenuto dell'accumulatore è maggiore o uguale al contenuto del registro B
6	End

instradare in sequenza le uscite dell'accumulatore e del registro B verso la ALU; quindi dispone l'assetto di questa unità per l'esecuzione di una addizione. Eseguita l'addizione, l'unità di controllo regola il trasferimento dei dati dalla ALU all'accumulatore.

Il funzionamento dell'unità di controllo è discusso nel Cap. 8.

2.3. Memoria centrale

La CPU e la sezione I/O di un microcomputer contengono diversi registri o buffer per la memorizzazione temporanea delle informa-

zioni digitali; tuttavia, la sede principale della memorizzazione dei dati è la *memoria centrale* (*main memory*). In confronto ai registri della CPU e ai buffer d'uscita della sezione I/O, gli elementi memorizzatori della memoria centrale sono più numerosi e normalmente più lenti.

La memoria centrale può comprendere svariati tipi di elementi memorizzatori; comunque, i tipi di gran lunga prevalenti nei microcomputer sono le ROM (*read-only memories: memorie a sola lettura*) e le RAM (*random-access memories: memorie ad accesso diretto,* o *non sequenziale* o *non ordinato*). In effetti, l'accesso è diretto per entrambi i tipi, e il *tempo di accesso* alle locazioni sostanzialmente il medesimo. La differenza risiede nel fatto che in una ROM i dati possono essere solamente letti, mentre in una RAM i dati possono essere letti oppure scritti. Inoltre, quando si spegne il computer, i dati di una RAM normalmente si perdono, mentre i dati di una ROM si conservano. Per la memorizzazione di informazioni permanenti, come la sequenza di controllo nella Tav. 2.1, sono quindi preferibili le ROM.

In generale, ROM e RAM in forma di circuito integrato possono immagazzinare un grande numero di bit: sono comuni ROM con capacità di 16384 bit (*16 k bit*) e RAM con capacità di 4096 bit (*4 k bit*).

Le celle binarie che compongono la memoria centrale di un microcomputer sono raggruppate in *parole* (*word*); ogni parola è identificata mediante la sua *locazione* o *posizione in memoria* (*memory location*), o *indirizzo* (*address*).

> **Esempio 2.8.** Un microcomputer con parole di 8 bit impiega una RAM da 2048 bit e una ROM da 4096 bit. La RAM è organizzata in 256 parole e la ROM in 512 parole. Ciascuna parola è identificata dall'indirizzo della sua posizione in memoria, che può essere uno qualunque dei 768 numeri interi da 0 fino a 767.

La realizzazione e il funzionamento della memoria centrale sono discussi in dettaglio nel Cap. 7.

2.4. Microprocessori

La tecnologia moderna consente di includere l'intero microcomputer di fig. 2.1 in un unico chip semiconduttore, e in effetti alcuni calcolatori sono stati costruiti in questo modo. Un approccio alterna-

tivo pone invece su un solo chip o su pochi chip la CPU e tutta o quasi tutta la sezione I/O. Tali chip vengono chiamati *microprocessori* (*microprocessor*) o *microelaboratori*.

Le caratteristiche e le limitazioni di un microcomputer sono determinate in larga misura dal microprocessore. Di conseguenza, la comprensione completa delle proprietà del microprocessore è essenziale per il progetto e per l'uso di un microcomputer.

Problemi

1. Quante linee si richiedono per collegare i tasti e il display nel calcolatore dell'Esempio 2.1, supponendo che non si impieghi la scansione dei tasti o del display? Supporre che ciascuna cifra del display richieda 4 linee.
2. Preparare una lista della sequenza di operazioni necessarie per eseguire l'addizione di due numeri di una cifra ciascuno nel calcolatore descritto nell'Esempio 2.1. Ogni riga della lista deve rappresentare una operazione singola.
3. Stimare i requisiti di memoria per il controller del traffico dell'Esempio 2.2.
4. Determinare i requisiti di memoria per il controller di temperatura dell'Esempio 2.5.
5. Preparare una lista delle operazioni equivalente alla sequenza della Tav. 2.1, ma costituita esclusivamente da operazioni singole.

3. FONDAMENTI DI PROGRAMMAZIONE

In un microcomputer la successione delle operazioni è stabilita da un *programma* (*program* o *code*), costituito da una lista di *istruzioni* e di *dati*. Una istruzione è memorizzata nella memoria centrale sotto forma di una o più *parole di istruzione* (*instruction word*); i dati sono memorizzati sotto forma di *parole di dati* (*data word*). Ogni istruzione comprende un *operatore* e alcuni *operandi*; l'operatore descrive l'*operazione* da eseguire sui dati indicati dall'operando o dagli operandi.

> **Esempio 3.1.** L'istruzione « ADD 4 » è un'abbreviazione per « Addiziona al contenuto dell'accumulatore il contenuto della locazione di memoria 4 ». La parola « ADD » è l'operatore, e il numero « 4 » è l'operando.

Il programma che stabilisce la successione delle operazioni è memorizzato nella memoria centrale del microcomputer; in sequenza, ogni istruzione viene *prelevata* (*fetched*) dalla memoria centrale e poi *eseguita* (*executed*).

> **Esempio 3.2.** L'istruzione della Tab. 2.1 corrispondente al contenuto del registro P pari a 3 (« Confronta nella ALU il contenuto dell'accumulatore con il contenuto del registro B »), è una *istruzione di confronto* che viene prelevata dalla memoria nella locazione 3. Essa è fatta eseguire dall'unità di controllo instradando le uscite dell'accumulatore e del registro B verso l'ingresso della ALU e disponendo questa per l'esecuzione di un confronto.

3.1. Linguaggio di macchina

Ogni istruzione è memorizzata nella memoria centrale del microcomputer secondo un codice numerico indicato come « *codice di istruzione* (*instruction code*) *in linguaggio di macchina* », che comprende l'operatore e l'operando o gli operandi.

> **Esempio 3.3.** In un certo microcomputer con parole di 8 bit, l'istruzione abbreviata con « ADD 4 » (vedi Esempio 3.1) è memorizzata in linguaggio di macchina con due parole di 8 bit. La prima parola è 00111000 ($=56_{10}$), che nel microcomputer considerato rappresenta l'operatore « ADD »; la seconda parola è 00000100 ($= 4_{10}$), che è l'operando « 4 ».

Il *corredo delle istruzioni* (*instruction set*) di un microcomputer comprende istruzioni aritmetiche, logiche, di caricamento, di memorizzazione, di salto, e istruzioni di ingresso e uscita. Alcuni esempi di queste istruzioni sono riportate qui di seguito; altre istruzioni sono discusse nei capitoli successivi.

Istruzioni aritmetiche

« Addiziona al contenuto dell'accumulatore il contenuto della locazione di memoria specificata ». Il risultato è memorizzato nell'accumulatore; il contenuto della locazione di memoria specificata resta inalterato.

Istruzioni logiche

Un esempio di istruzione logica è l'istruzione di confronto dell'Esempio 3.2.

Istruzioni di caricamento (Loading)

« Carica nell'accumulatore il contenuto della locazione di memoria specificata (*Load accumulator*). Se l'operatore di questa istruzione

è seguito dall'operando 00000110 (= 6_{10}), allora il contenuto della locazione di memoria 6 è caricato nell'accumulatore; il contenuto di questa locazione di memoria resta inalterato.

« Carica nell'accumulatore il numero specificato (*Load immediate*). Se l'operatore di questa istruzione è seguito dall'operando 00001111 (= 15_{10}), allora il numero 15 è caricato nell'accumulatore.

Istruzioni di memorizzazione (Storing)

« Memorizza il contenuto dell'accumulatore nella locazione di memoria specificata ». Se l'operatore di questa istruzione è seguito dall'operando 00001101 (= 13_{10}), allora il contenuto dell'accumulatore è memorizzato nella locazione di memoria 13. Il contenuto dell'accumulatore resta inalterato.

Istruzioni di salto (Jump)

« Salta alla locazione di memoria specificata » (*salto incondizionato*: *Unconditional jump*). Il contenuto del contatore di programma (registro P) è sostituito dall'indirizzo della locazione specificata; l'istruzione successiva è prelevata quindi da quella locazione.

« Salta alla locazione di memoria specifica se il contenuto dell'*indicatore* (*flag*) specificato è il numero binario 1 » (*salto condizionato*: *Conditional jump*). Il flag, che è un registro da 1 bit, può indicare un riporto aritmetico, un prestito aritmetico, un *overflow* (*superamento di capacità o trabocco*), oppure può essere il flag del segno o dello zero, o il flag di qualche altra condizione (*condition flag*).

Istruzioni di ingresso e di uscita

« Ricevi i dati dal dispositivo di ingresso specificato ». I dati provenienti dal dispositivo di ingresso specificato sono caricat nel registro I/O attraverso il multiplexer di ingresso.

« Manda i dati al dispositivo di uscita specificato ». I dati del registro I/O sono caricati nel buffer del dispositivo di uscita specificato attraverso il multiplexer di uscita.

3.2. Linguaggio assemblativo

Nel paragrafo precedente si sono illustrate alcune istruzioni in linguaggio di macchina. Un programma può essere costituito da centinaia di tali istruzioni, e deve essere preparato con cura meticolosa. L'impiego delle istruzioni in linguaggio di macchina trova il suo limite nell'incapacità di un essere umano di manipolare lunghe strisce di cifre binarie senza commettere errori; volga l'esempio dell'istruzione « Addiziona al contenuto dell'accumulatore il contenuto della locazione di memoria 4 » (cfr. Esempi 3.1 e 3.3), che può risultare codificata in linguaggio di macchina come 0011100000000100. Il numero di errori commessi nella trascrizione di tali istruzioni può essere considerevolmente ridotto ricorrendo ad una *abbreviazione mnemonica* come « ADD 4 », come è stato fatto negli Esempi 3.1 e 3.3. Un linguaggio di programmazione nel quale la codificazione in linguaggio di macchina di ogni istruzione è sostituito da una abbreviazione mnemonica, viene indicato come *linguaggio assemblativo* (*assembly language*) o *linguaggio simbolico* (*symbolic language*).

Ovviamente, per il funzionamento del microcomputer si richiede che un programma scritto in un linguaggio assemblativo venga tradotto in un corrispondente programma in linguaggio di macchina; questa traduzione però viene eseguita da un programma *assemblatore* (*assembler*).* L'assembler può essere *residente* nella memoria centrale del microcomputer (*resident assembler*), oppure può essere situato nella memoria di un altro computer, normalmente di maggiori dimensioni (*cross-assembler*).

> **Esempio 3.4.** Un programma per un microcomputer, stilato in un linguaggio assemblativo, viene perforato su un nastro mediante un perforatore di nastro, che è una macchina comandata da una tastiera simile a quella di una telescrivente. Il nastro così ottenuto viene passato ad un lettore di nastro perforato che agisce come dispositivo di ingresso di un grosso computer, nel quale viene introdotto anche il programma cross-assembler, scritto in un linguaggio comprensibile al computer. Sotto la direzione del programma cross-assembler, il computer traduce il programma scritto in linguaggio assem-

---

* Maggiori dettagli sui programmi assemblatori sono forniti nel Cap. 9.

blativo nel corrispondente programma scritto nel linguaggio di macchina del microcomputer; lo perfora quindi su un nastro di carta, che viene poi fatto leggere al microcomputer.

Macroistruzioni

Un programma assemblatore, oltre a tradurre le istruzioni dal linguaggio assemblativo al linguaggio di macchina, può servire anche ad altri scopi. Per esempio, se un gruppo abbastanza esteso di istruzioni si ripete più volte nel programma, esso può essere definito come una *macroistruzione*, che viene poi usata al posto del gruppo; in questo caso, nella fase di assemblaggio del programma, l'assemblatore sostituisce alla macroistruzione il gruppo di istruzioni corrispondente, come è necessario.

Esempio 3.5. Un gruppo di 20 istruzioni interviene identico 10 volte nello stesso programma; per risparmiare tempo e fatica, viene introdotta la macroistruzione MAC1, definita nel modo seguente:

La macroistruzione MAC1 viene poi scritta in 10 differenti posizioni del programma. In fase di assemblaggio, ogni volta che incontra una delle 10 macroistruzioni MAC1, l'assemblatore la sostituisce con il corrispondente gruppo di 20 istruzioni in linguaggio di macchina.

L'uso delle macroistruzioni non altera quindi il programma definitivo in linguaggio di macchina, ma consente di evitare di scrivere più volte nel programma in linguaggio assemblativo la stessa lunga sequenza di istruzioni. Uno spiacevole effetto laterale prodotto dall'uso delle macroistruzioni è che il programma in linguaggio assemblativo può apparire ingannevolmente breve. Di conseguenza, ogni volta che si usano macroistruzioni, si deve avere cura di stimare la grandezza del programma, ossia l'ingombro di memoria.

Esempio 3.6. Un programma in linguaggio assemblativo comprende 30 istruzioni, 12 delle quali sono macroistruzioni MAC1 definite nell'Esempio 3.5. Di conseguenza, il programma prodotto dall'assemblatore è costituito da 258 istruzioni in linguaggio di macchina, che superano le 256 locazioni di memoria disponibili in quel particolare microcomputer.

3.3. Linguaggi di programmazione ad alto livello

Per il programmatore che deve scrivere o comprendere un programma, l'uso di un linguaggio assemblativo rappresenta un notevole progresso rispetto all'uso del linguaggio di macchina. Tuttavia, il lavoro di stesura del programma resta ancora notevole, ed anche un programma semplice come quello illustrato nell'Esempio 2.6, può richiedere decine di istruzioni.

Il lavoro del programmatore è stato alleviato ulteriormente con l'introduzione dei *linguaggi di programmazione ad alto livello.* Questi linguaggi di programmazione combinano diverse istruzioni in un'unica *frase* o *espressione* (*statement*); un programma è allora costituito da una successione di tali frasi.

Esempio 3.7. In ALGOL, che è un linguaggio di programmazione ad alto livello, la frase $Y \leftarrow Z$ significa: « Assegno ad Y il valore di Z ». Uno *statement* più complesso è il seguente: IF $V > W$ THEN $X \leftarrow Y - 1$ ELSE $X \leftarrow Y + 1$.

Un programma scritto in un linguaggio di programmazione ad alto livello normalmente è molto più corto di un programma scritto in un linguaggio assemblativo, e può risultare anche più corto della presentazione originale del problema.

Esempio 3.8. Il programma dell'esempio 2.6 può essere scritto in ALGOL come segue:

BEGIN
$A \leftarrow THERMOSTAT$;
$B \leftarrow M$;
IF $A < B$ THEN $HEATER \leftarrow 1$ ELSE $HEATER \leftarrow 0$;
END
END.

Un linguaggio di programmazione ad alto livello è tradotto in un linguaggio di macchina da un programma *compilatore* (*compiler*). Il programma compilatore esamina ogni statement, assegna le locazioni di memoria per la memorizzazione delle variabili e delle costanti, e genera le istruzioni in linguaggio di macchina.

Una questione importante relativa all'impiego del compilatore è la sua efficienza. Poiché il compilatore è solamente un programma ed agisce quindi in maniera meccanica, il programma in linguaggio di macchina da esso generato normalmente comprende un numero di istruzioni considerevolmente più elevato che il programma in linguaggio di macchina scritto da un programmatore esperto, e quindi occupa in memoria uno spazio notevolmente maggiore. Questa inefficienza può risultare critica per un piccolo microcomputer più che per un grosso computer, sebbene possa talvolta risultare importante anche per quest'ultimo. Per questa ragione al programmatore spesso viene data la possibilità di combinare in un singolo programma un linguaggio assemblativo e un linguaggio di programmazione ad alto livello.

3.4. Sottoprogrammi

Abbiamo già visto che l'uso delle macroistruzioni semplifica la scrittura di un programma in linguaggio assemblativo senza alterare il programma definitivo in linguaggio di macchina. Quando in un programma si presenta più volte una sequenza di istruzioni o di statement, è preferibile che tale sequenza faccia parte del programma in linguaggio di macchina solo una volta, e venga poi *richiamata* (*called*) ogni volta che serve. Una simile sequenza di istruzioni viene indicata come *sottoprogramma* (*subroutine* o *procedure*). Esso può essere scritto nel linguaggio di macchina, in un linguaggio assemblativo o in un linguaggio di programmazione ad alto livello. Se ricorre numerose volte in un programma, un sottoprogramma dà luogo ad un considerevole risparmio di spazio di memoria.

Esempio 3.9. In un programma si deve usare 100 volte la formula ella fornisce una radice di un'equazione di secondo grado, $ROOT = (-B + \sqrt{B^2 - 4AC})/2A$, ogni volta con valori diversi per A, B e C. Per il calcolo di $ROOT$ conviene

allora ricorrere ad un sottoprogramma, che in ALGOL si può scrivere come segue:

PROCEDURE *ROOT* (*A*, *B*, *C*);
ROOT $\leftarrow$ 0;
$D \leftarrow B \times B - 4 \times A \times C$;
IF $D \geqq 0$ THEN *ROOT* $\leftarrow (-B + \mathrm{SQRT}(D))/(2 \times A)$;
PRINT (*A*, *B*, *C*, *D*, *ROOT*);
END;

Si noti che SQRT (*D*) corrisponde ad un sottoprogramma separato per il calcolo di $\sqrt{D}$, che interviene quando $D \geq 0$. Quando $D < 0$, il testo stampato mostrerà $ROOT = 0$ e un valore negativo per *D*, indicando in tal modo che il valore corretto per *ROOT*, che in questo caso risulta complesso, non può essere ottenuto con questo sottoprogramma elementare.

Il risparmio di spazio di memoria conseguito ricorrendo ad un sottoprogramma è in parte ridotto dalla spesa (*overhead*) in spazio di memoria e in tempo di esecuzione dovuta alla presenza dei *collegamenti di sottoprogramma* richiesti per evitare nel sottoprogramma e per uscire da esso. I collegamenti di sottoprogramma sono discussi nel Cap. 9.

3.5. Diagrammi di flusso

Uno strumento importante per lo sviluppo di un programma è il *diagramma di flusso delle operazioni* (*flow chart*). A meno che il programmatore non sia molto esperto e il problema veramente semplice, è consigliabile cominciare tracciando un flow chart basato sulla presentazione originale del problema e procedere poi alla scrittura del programma. Un flow chart in ogni caso è costituito da una combinazione degli elementi mostrati in fig. 3.1.

Esempio 3.10. In fig. 3.2 è mostrato il flow chart del sottoprogramma dell'Esempio 3.9, relativo al calcolo di una radice di una equazione di secondo grado. Esso è costituito da sette

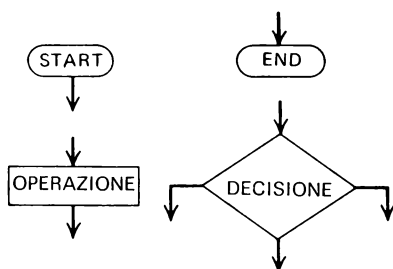



FIG. 3.1. - Elementi di *flow chart.*

blocchi: un blocco START, uno END, quattro blocchi OPERAZIONE e un blocco DECISIONE.

Durante l'esecuzione di un programma, ogni elemento di un flow chart può essere attraversato un numero finito qualsiasi di volte, ad eccezione dei blocchi START e END, che possono essere attraversati una volta soltanto. Una struttura di flow chart molto importante, relativa alla esecuzione ripetuta di un segmento di programma, è il *ciclo* (*loop* o *DO loop*).

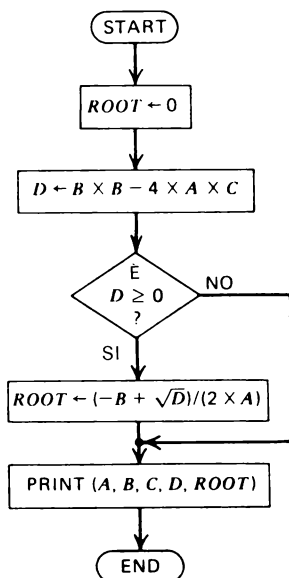



FIG. 3.2. - *Flow chart* per il calcolo di una radice di una equazione di secondo grado.

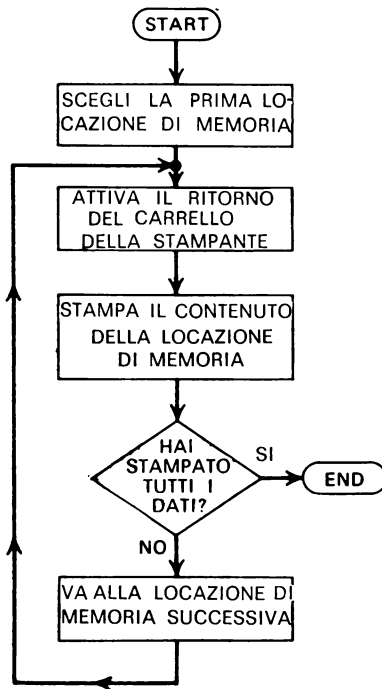



FIG. 3.3. - *Flow chart* di un programma per la stampa dei contenuti di un gruppo di locazioni di memoria.

Esempio 3.11. La fig. 3.3 mostra il flow-chart di un programma che stampa il contenuto di un blocco di locazioni di memoria. Scelta la prima locazione di memoria da stampare, il programma stampa il contenuto di ogni successiva locazione di memoria percorrendo il *loop* del *flow-chart* finché tutti i dati non sono stati stampati.

Problemi

1. Il codice di operazione in linguaggio di macchina per l'operatore dell'istruzione « load accumulator » (« carica l'accumulatore ») è 00001000, e per l'istruzione « load immediate » è 0001001. Il contenuto della locazione di memoria 11_{10} è 5_{10}. Qual'è il contenuto dell'accumulatore dopo l'esecuzione di ciascuna delle seguenti istruzioni:

   0000100000001011, 0000100100001011, e 0000100111111111?

2. Stabilire un insieme di abbreviazioni mnemoniche e scrivere un programma in questo linguaggio assemblativo per il sistema di riscaldamento dell'Esempio 2.6.
3. Riassumere i vantaggi e gli eventuali svantaggi di un programma scritto in linguaggio di macchina, in un linguaggio assemblativo e in un linguaggio di programmazione ad alto livello.
4. Discutere i vantaggi e i possibili svantaggi di un sottoprogramma in confronto ad una macroistruzione.
5. Tracciare il flow chart e delineare un sottoprogramma per il calcolo della media di quattro numeri A, B, C e D.
6. Preparare il flow chart del funzionamento del controller di temperatura descirtto nel'Esempio 2.5.
7. Quante locazioni di memoria sono richieste nel programma dell'Esempio 3.6 se le macroistruzioni sono sostituite da un sottoprogramma? Includere una istruzione di chiamata per ogni entrata nel sottoprogramma e una istruzione di ritorno per ogni uscita da questo.

4. INGRESSO E USCITA

La comunicazione fra il computer e il mondo esterno avviene attraverso i *dispositivi I/O* (*input/output*: *ingresso/uscita*) o *unità periferiche*. Questo capitolo discute la forma delle istruzioni usate per l'azionamento di questi dispositivi, la struttura e il funzionamento della sezione I/O, l'interconnessione dei dispositivi I/O, l'interruzione del programma e l'accesso diretto alla memoria.*

4.1. Istruzioni di ingresso e di uscita

Le istruzioni I/o di un microcomputer, già introdotte nel par. 3.1, hanno la forma « Carica nel registro I/O i dati dell'unità d'ingresso specificata » e « Carica i dati del registro I/O nel buffer dell'unità d'uscita specificata ». Queste istruzioni si incaricano del trasferimento di tutte le informazioni, comprese quelle di controllo. Un'*informazione di controllo* può essere il segnale di *pronto* (*ready*) di una stampante elettrica, il segnale *fine delle schede* (*end of card*) del lettore di schede perforate, il segnale *pronto per un'altra scheda* (*ready for next card*) di un perforatore di schede, il comando *riavvolgi* (*rewind*) ad una unità lettrice di nastro magnetico, il segnale *riavvolgimento completato* (*rewind completed*) oppure il segnale *fine del nastro* (*end of tape*) di una unità a nastro magnetico, o altri ancora.

---

* Una descrizione dettagliata dei dispositivi periferici si può trovare in Bibliografia 2.

In vario grado i microcomputer consentono o richiedono l'intervento del programmatore nella manipolazione delle informazioni di controllo che possono essere incluse nelle istruzioni I/O.

4.2. Sezione I/O

I componenti fondamentali della *sezione I/O*, detta anche *controller I/O*, sono il registro I/O, i multiplexer I/O e i buffer di uscita (fig. 2.2). Questo paragrafo presenta alcuni particolari relativi al funzionamento di questi circuiti.

Registro I/O

Il *registro I/O* si incarica dello scambio delle informazioni fra la sezione I/O e la CPU del microcomputer. Quando la capacità di questo registro è insufficiente per la trasmissione simultanea dei dati, delle informazioni di controllo e di quelle per l'*identificazione* della *periferica* (*device identification*), la trasmissione ha luogo in blocchi consecutivi.

> **Esempio 4.1.** Un microcomputer con parole lunghe 8 bit ha 16 dispositivi I/O, alcuni dei quali richiedono informazioni di controllo di 4 bit. La sezione I/O e il registro I/O, devono quindi manipolare 4 bit del codice di identificazione del dispositivo e 4 bit di informazione di controllo, in aggiunta agli 8 bit dei dati. Tutto questo viene organizzato in due gruppi consecutivi: il primo gruppo di 8 bit è costituito da 4 bit di identificazione della periferia e da 4 bit dell'informazione di controllo, mentre il secondo gruppo è costituito da 8 bit di dati.

Una caratteristica importante del registro I/O è che esso deve assicurare il flusso bidirezionale delle informazioni. Per un bit questo risultato può essere ottenuto con il circuito di fig. 4.1, nel quale l'elemento di memoria è costituita da un flip-flop *D*. Tipico di questo flip-flop è che il valore *futuro* dell'uscita *Q* coincide con il valore *presente dell'ingresso D*, dove *presente* e *futuro* stanno per prima e dopo

l'impulso applicato all'ingresso *C* dal *clock* (orologio, o pilota di cadenza). Vengono usati circuiti logici *tristate* (a tre stati), che presentano in uscita un circuito aperto quando l'ingresso *ENABLE*, spesso indicato anche con $\overline{INHIBIT}$, è al livello logico 0.* L'ingresso *ACTIVATE* consente il flusso delle informazioni, mentre l'ingresso *DIRECTION* ne stabilisce il senso, se verso destra o verso sinistra.

Multiplexer e buffer

Come è detto nel par. 2.1, un microcomputer può comprendere diversi *multiplexer*. La struttura di un multiplexer varia a seconda che il flusso delle informazioni debba avvenire in un solo senso o in entrambi i sensi. Inoltre, nel caso più semplice di un multiplexer unidirezionale (fig. 2.2), la sua struttura dipende dal fatto che esso si trovi in ingresso o in uscita.

In fig. 4.2 è mostrato lo schema circuitale di un multiplexer unidirezionale di ingresso. Esso gestisce tre dispositivi di ingresso ciascuno con parole di 4 bit. La confluenza dei dati è ottenuta mediante

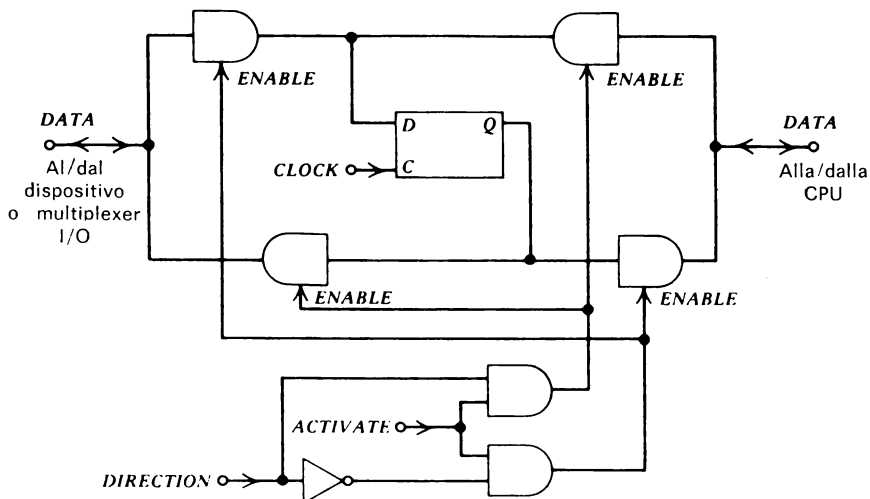



FIG. 4.1. - Flusso bidirezionale dei dati in un registro I/O da 1 bit.

* Maggiori dettagli sui circuiti logici si possono trovare in Bibliografia 1.

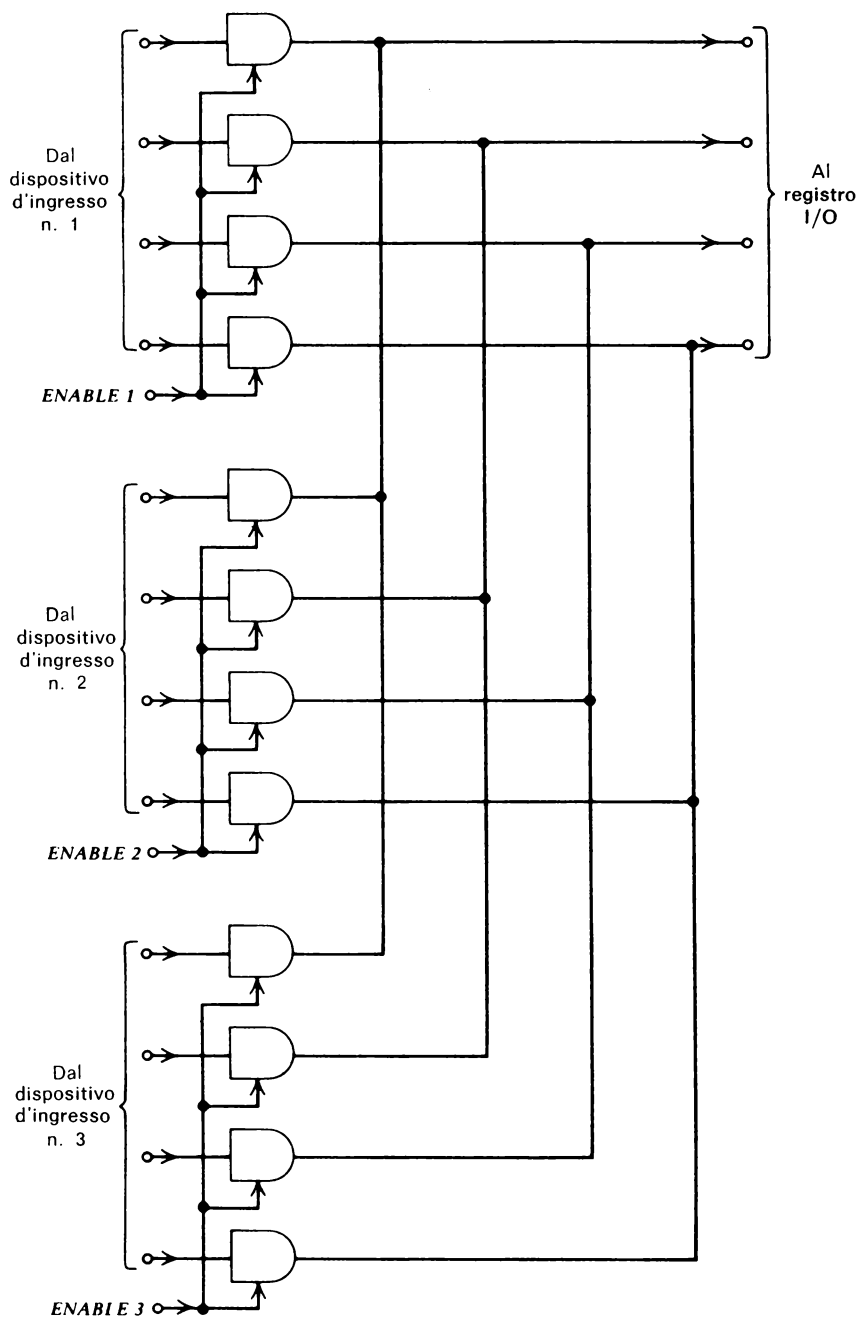



FIG. 4.2. - Multiplexer d'ingresso unidirezionale realizzato con circuiti logici *tristate.*

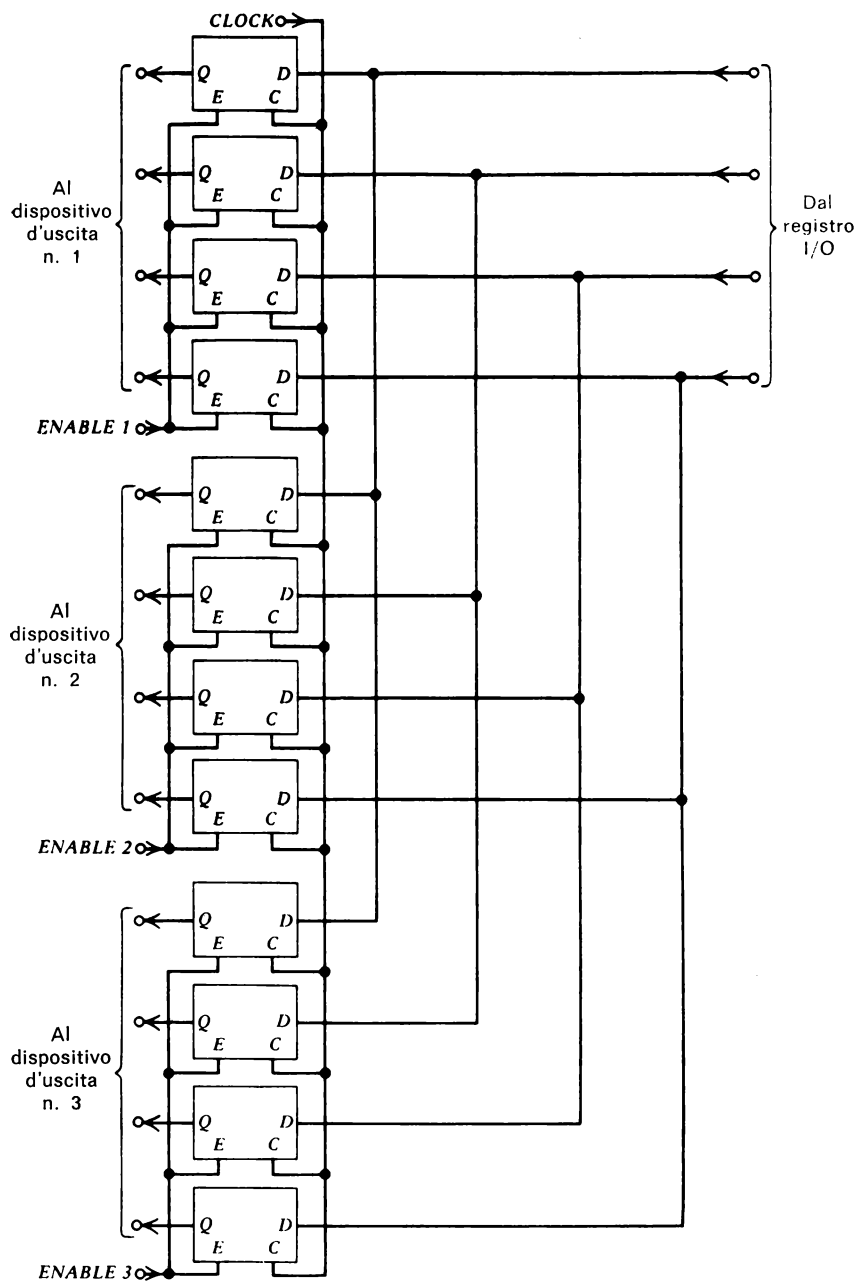



FIG. 4.3. - Multiplexer d'uscita unidirezionale e buffer d'uscita realizzati con flip-flop D-E.

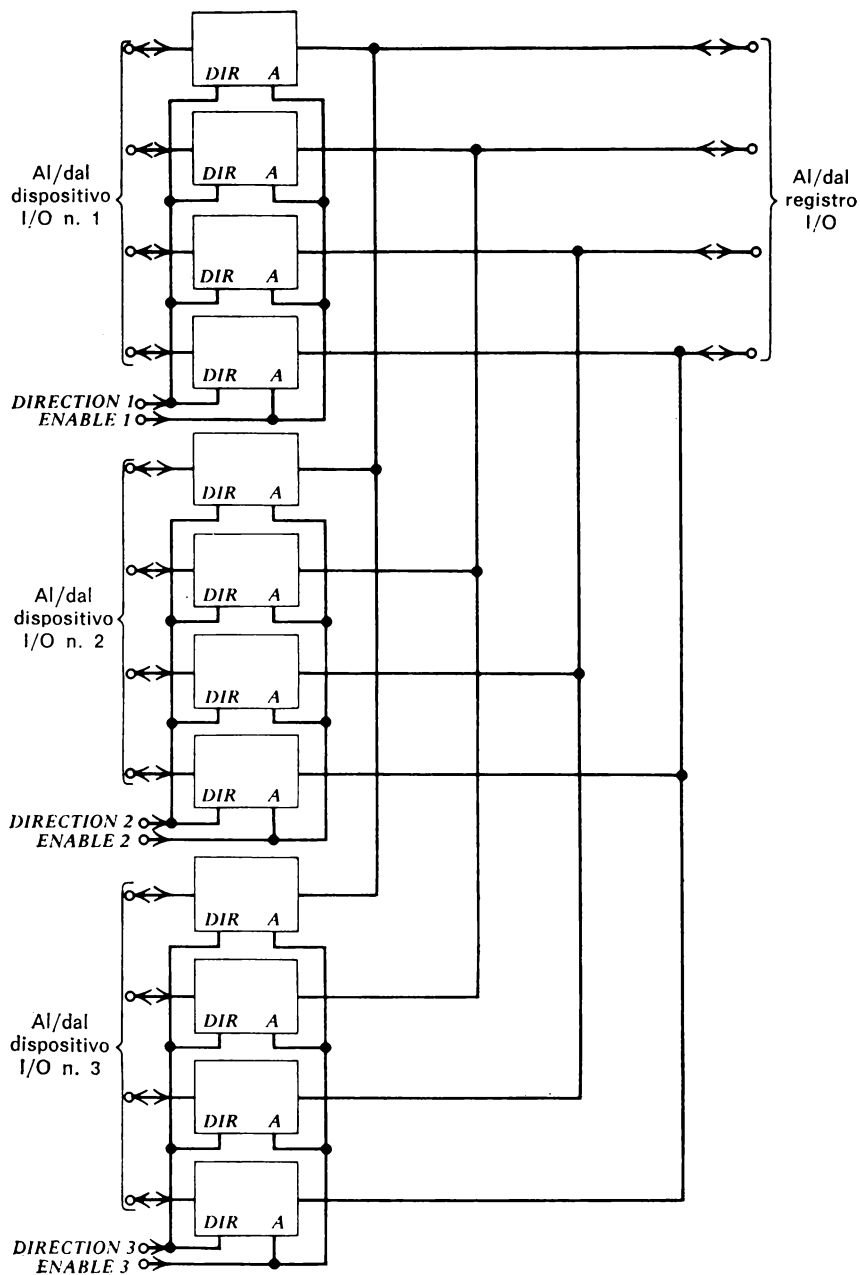



FIG. 4.4. - Multiplexer I/O bidirezionale. Ciascuno dei dodici blocchi rappresenta il cricuito di fig. 4.1, dove le linee *DIRECTION* sono marcate con *DIR*, le linee *ACTIVATE* sono marcate con *A*, e e linee *DATA* non sono marcate.

circuiti logici con uscite *tristate*, attivati da uno dei tre ingressi *ENABLE*; in ogni istante, solo un ingresso *ENABLE* può assumere il valore logico 1.

In fig. 4.3 è mostrato invece lo schema circuitale di un multiplexer unidirezionale di uscita, comprendente i *buffer di uscita*, formati da *flip-flop D-E*. Per questo tipo di flip-flop, il valore futuro dell'uscita *Q* coincide con il valore presente dell'ingresso *D* se l'ingresso *E* (*ENABLE*) si trova a livello logico 1; il valore dell'uscita resta invece immutato, indipendentemente dal valore di *D*, se l'ingresso *E* è a livello logico 0. Gli ingressi *ENABLE* (*E*) relativi ad uno stesso dispositivo sono collegati in parallelo.

> **Esempio 4.2.** Nella sezione I/O del controller del traffico dell'Esempio 2.2, i dispositivi di ingresso sono separati dai dispositivi d'uscita; si possono quindi usare multiplexer simili a quelli delle figg. 4.2 e 4.3.

Quando le linee che collegano i dispositivi d'ingresso e d'uscita non sono separate, si deve usare un multiplexer bidirezionale. In fig. 4.4 è riportato il circuito di un multiplexer di questo tipo, ottenuto ripetendo il circuito base di fig. 4.1. Tutti gli ingressi *DIRECTION* (*DIR*) relativi ad uno stesso dispositivo sono collegati in parallelo, come anche gli ingressi *ACTIVATE* (*A*). La selezione di un dispositivo avviene ponendo a 1 i suoi ingressi *ENABLE*; in ogni istante, solo ad una delle linee *ENABLE* è consentito di essere a livello binario 1.

I dati e l'informazione di controllo relativi ad un certo dispositivo possono essere trattati insieme se fluiscono nello stesso senso; nel caso contrario, possono essere trattati come appartenenti a due dispositivi differenti con opposto senso di flusso dei dati. La situazione è simile quando l'informazione di controllo deve fluire nei due sensi (funzionamento *handshake*: a stretta di mano).

In fig. 4.5 è tracciata l'architettura complessiva di una sezione I/O. Essa comprende un *multiplexer ausiliario*, che consente di trasferire le informazioni di identificazione del dispositivo I/O al *registro ID* e permette inoltre lo scambio dei dati e delle informazioni di controllo fra il *registro dei dati I/O* e il registro I/O. Questo multiplexer non è necessario nei microcomputer che hanno *linee bus* separate per i dati e gli indirizzi. Il contenuto del registro ID è decodificato dal *decodificatore ID*. In molti casi, specie quando sono presenti numerosi dispositivi I/O, può risultare più economico porre questi decodificatori,

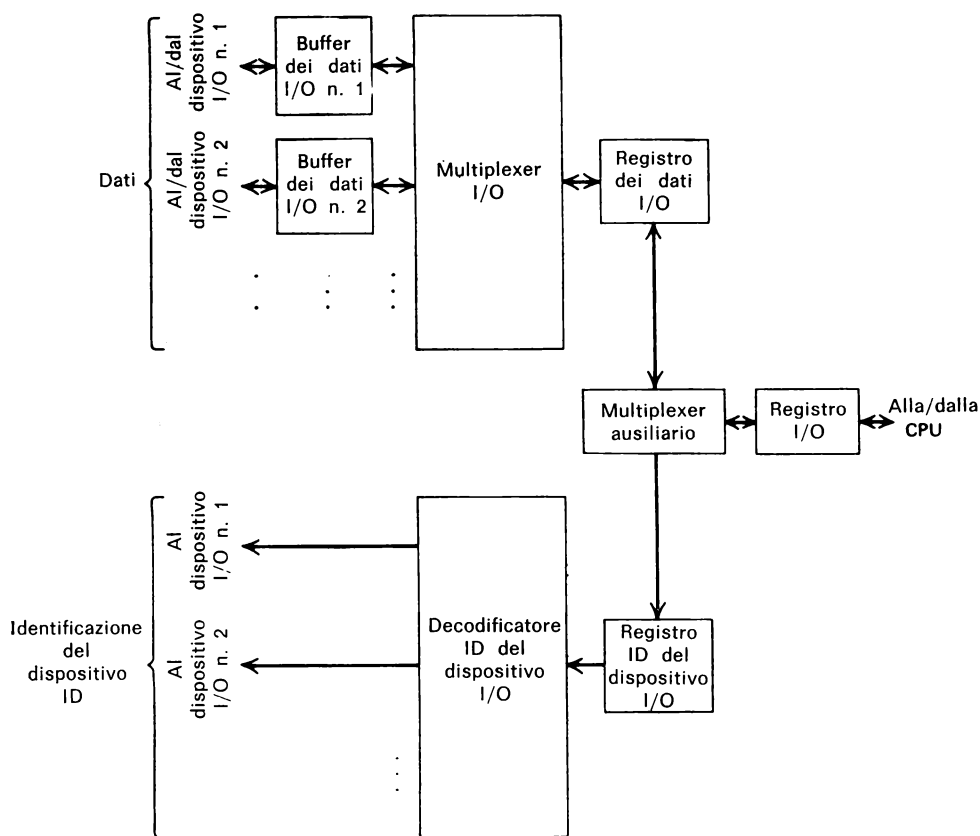



FIG. 4.5. - Schema complessivo semplificato di una sezione I/O.

come anche i *buffer dei dati I/O*, in corrispondenza dei dispositivi I/O piuttosto che nel microcomputer. La decisione deve essere basata su diversi fattori, come il numero dei dispositivi I/O, la lunghezza della parola, la posizione dei dispositivi I/O rispetto al microcomputer, e i circuiti integrati disponibili.

4.3. Interruzioni

Finora elaborazione e operazioni I/O risultano attività distinte ed indipendenti, poiché le unità I/O vengono fatte funzionare sotto il controllo del programma (*polling*). In un altro modo di funziona-

mento, un'unità I/O può richiedere il servizio inviando al microcomputer un segnale di *richiesta di interruzione* (*interrupt request*).

> **Esempio 4.3.** Il funzionamento del controller della temperatura dell'Esempio 2.5 viene diretto dalla tastiera di una telescrivente, stampando la temperatura desiderata e attivando il ritorno del carrello. In questo modo si invia al microcomputer un segnale di interruzione, richiedendo l'elaborazione delle informazioni.

Ricevuto un segnale di richiesta di interruzione, il microcomputer completa l'istruzione in corso di esecuzione e chiama poi un *sottoprogramma di interruzione* (*interrupt subroutine*). In taluni microcomputer, una apposita circuiteria può scegliere fra diversi sottoprogrammi d'interruzione (*vectored interrupt*). Eseguito il servizio del dispositivo di ingresso, il controllo passa di nuovo dal sottoprogramma di interruzione alla normale sequenza delle istruzioni. Richieste d'interruzione possono essere avanzate anche dai dispositivi d'uscita. Certi dispositivi d'uscita, come il perforatore di nastro, possono attendere indefinitamente che il microcomputer fornisca il carattere successivo e possono quindi utilizzare sia il *polling* sia l'*interrupt*; invece altri dispositivi d'uscita, come un'unità a nastro magnetico, talvolta richiedono il servizio entro breve tempo, per cui devono ricorrere all'*interrupt* per richiamare l'attenzione.

> **Esempio 4.4.** Un registratore di nastro magnetico a cassetta, che costituisce un'unità periferica a basso costo, viene fatto partire dal microcomputer mediante un comando di *start*. Raggiunta la velocità di funzionamento normale, esso registra 1000 caratteri al secondo.
>
> Ogni volta che è pronto per registrare un carattere, il registratore a cassetta invia al microcomputer un segnale di interruzione richiedendo il carattere successivo. Per un corretto funzionamento dell'unità a nastro, il microcomputer deve servire questo *interrupt* entro un intervallo di tempo di circa 1 millisecondo.

Molti microcomputer sono provvisti di un sistema di *interruzione multipla* (*multiple interrupt*), accessibile a diversi dispositivi I/O, ai quali nell'accesso vengono assegnate *priorità* (*priority*) uguali o differenti.

4.4. Accesso diretto alla memoria

L'accesso diretto alla memoria (*direct memory access*: DMA), evitando la sezione I/O, consente lo scambio di dati ad alta velocità direttamente fra la memoria centrale e un dispositivo perferico. Normalmente il funzionamento della sezione I/O e della CPU viene sospeso durante il DMA, e il controllo della memoria viene assegnato ad un apposito circuito logico esterno. La necessità di questo circuito ausiliario diminuisce però notevolmente il vantaggio della maggiore velocità; per cui il DMA viene installato soprattutto quando si richiede il trasferimento ad alta velocità di grandi blocchi di dati.

Problemi

1. In fig. 4.1, determinare il senso di flusso dei dati quando l'ingresso *DIRECTION* si trova a livello binario 1.
2. Tracciare il diagramma temporale dei segnali del multiplexer bidirezionale in fig. 4.4 quando esso aziona sequenzialmente un dispositivo di ingresso e due dispositivi d'uscita.
3. Estendere il diagramma temporale del Problema 2 includendo i segnali *DIRECTION* e *ACTIVATE* del registro I/O.
4. Tracciare un circuito logico che mostri i dettagli della sezione I/O di fig. 4.5. Supporre che i dispositivi periferici siano 3 e che la lunghezza delle parole sia di 4 bit.
5. Tracciare il diagramma temporale dei segnali del circuito del Problema 4 quando esso aziona sequenzialmente un dispositivo di ingresso e due dispositivi d'uscita.
6. I buffer dei dati I/O e il decodificatore ID possono essere disposti nel microcomputer oppure in corrispondenza del dispositivo I/O. Considerare i fattori che influenzano la decisione.
7. Stabilire un ragionevole ordine di priorità fra una telescrivente elettrica, un registratore di nastro a cassetta e un perforatore di nastro. Tracciare un circuito che gestisca l'interrupt multiplo e che fornisca inoltre al microcomputer l'identificazione del dispositivo I/O che richiede l'interrupt.

5. OPERAZIONI ARITMETICHE

Questo capitolo descrive i vari *sistemi di numerazione* utilizzati in un microcomputer, la *rappresentazione in virgola mobile* e l'*aritmetica in virgola mobile*. In esso si discute anche della codificazione dei caratteri dell'alfabeto e degli altri simboli.

5.1. Sistemi di numerazione

L'ordinario sistema di numerazione impiega la *base* dieci e la *notazione posizionale*; questo significa che ciascuna cifra di una parola di più cifre possiede un *peso* costituito da una potenza in base dieci. Per esempio, il numero decimale 4956 può essere espresso come la somma di coefficienti con peso:

$$4956 = 4 \times 10^3 + 9 \times 10^2 + 5 \times 10^1 + 6 \times 10^0.$$

La base decimale non è l'unica base in cui si possono esprimere i numeri. Per esempio, in conseguenza della natura binaria dei circuiti di un computer, ha trovato largo favore il sistema di numerazione binario. Questo sistema usa solamente i segni 0 e 1, per cui $1 + 0 = 0 + 1 = 1$; $1 + 1 = 10$ (si legge: zero con riporto di uno); $10 + 1 = 11$, e così via. Le regole per le operazioni aritmetiche sono simili a quelle del sistema decimale, solo si deve tener conto che il sistema di numerazione binario usa un insieme di due cifre binarie (0, 1) al posto di un insieme di dieci cifre decimali (0, 1, ..., 9).

In generale, ogni numero N_b costituito da n cifre ed assegnato in una certa base b può essere espresso come somma dei suoi coefficienti

pesati:

$$N_b = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_1b^1 + a_0b^0 = \sum_{i=0}^{n-1} a_ib^i \qquad (5.1)$$

Nella notazione posizionale i pesi non vengono indicati esplicitamente, e si scrive:

$$N_b = a_{n-1}\, a_{n-2} \ldots a_1a_0 \qquad (5.2)$$

Numeri binari

I numeri binari sono numeri espressi nella base 2, e per questo spesso sono indicati con il pedice 2. Per esempio, 11_2 esprime un numero in notazione binaria equivalente al numero decimale 3_{10}. Poiché entrambi i sistemi di numerazione, binario e decimale, sono largamente usati, verrà ora descritto in dettaglio la conversione fra questi due sistemi di numerazione. Infatti, un numero in una data base può essere espresso in un'altra base da un valore numerico equivalente mediante l'applicazione di un opportuno *algoritmo di conversione.*

Conversione da binario a decimale

Vengono qui presentati due metodi per la conversione da binario a decimale. Uno di questi è basato sull'equazione 5.1 ed è illustrato dagli Esempi 5.1 e 5.2.

Esempio 5.1. Convertire in base 10 il numero intero binario $N_2 = 111001_2$.

Secondo l'equazione 5.1, N_2 rappresenta la somma:

$$N_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Di conseguenza:

$$N_2 = (32 + 16 + 8 + 1)_{10} = 57_{10}.$$

In modo analogo si possono convertire i numeri frazionari, ricordando che il valore di ciascuna cifra, o *bit, dopo il punto* * *binario*

---

* È necessario considerare che secondo l'uso anglosassone la parte frazionaria viene preceduta da un punto e non da una virgola (*N.d.T.*).

cambia secondo un fattore 1/2. L'equazione 5.1 può così essere estesa ad includere *bit* frazionari posti alla destra del punto binario. Assegnando a questi coefficienti i pedici $a_{-1}, a_{-2}, \ldots, a_{-m}$, si ha:

$$N_2\,(\text{frazionario}) = a_{-1}\,2^{-1} + a_{-2}\,2^{-2} + \ldots + a_{-m+1}\,2^{-m+1} + a_{-m}\,2^{-m}$$

$$= \sum_{i=-m}^{-1} a_i 2^i. \tag{5.3}$$

Esempio 5.2. Convertire in base 10 il numero binario frazionario $N_2 = 0.11001_2$.

Si ha:

$$N_2\ (\text{frazionario}) = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} +$$

$$+ 1 \times 2^{-5} = (0.5 + 0.25 + 0.031125)_{10} = 0.78125_{10}$$

Un altro modo per convertire un numero binario in numero decimale, applicabile solamente ai numeri interi, è il metodo *double-dabble.* Esso è basato sulla seguente rappresentazione a nido dei numeri binari:

$$N_2 = \{[(a_{n-1} \cdot 2 + a_{n-2}) \cdot 2 + {}_{n-3}] \cdot 2 + \ldots + a_1\} \cdot 2 + a_0 \tag{5.4}$$

L'equazione 5.4 indica il procedimento per la conversione: il bit più significativo (*msb*: *most significant bit*) viene raddoppiato e poi sommato al coefficiente del bit successivo, che può essere 0 oppure 1; il risultato viene raddoppiato di nuovo, sommato, e così via; il procedimento termina quando si somma infine il bit meno significativo (*lsb*: *least significant bit*).

Esempio 5.3. Per convertire il numero binario 1001101 nel suo equivalente decimale ricorrendo al metodo *double-dabble* si procede come illustrato in fig. 5.1.

Conversione da decimale a binario

Un metodo per la conversione da decimale a binario con mezzi manuali, adatto quindi solo per numeri piccoli, interi o frazionari, è

fondato sul riconoscimento delle potenze di 2 contenute nel numero decimale. Si parte sottraendo la più elevata potenza di 2 contenuta nel numero decimale da convertire e si continua finché la conversione non è stata completata.

Esempio 5.4. Per convertire 167_{10} in numero binario, si comincia notando che 2^7 è la più elevata potenza di 2 contenuta nel numero da convertire; si fa poi $167 - 2^7 = 167 - 128 = 39$; si fa quindi $39 - 2^5 = 7 = 111_2$, e così via. Il risultato è allora $167 = 2^7 + 2^5 + 2^2 + 2^1 + 2^0$, ossia $167_{10} = 10100111_2$.

Questo metodo risulta poco pratico per numeri grandi. Un altro procedimento di conversione da decimale a binario fa ricorso a suc-

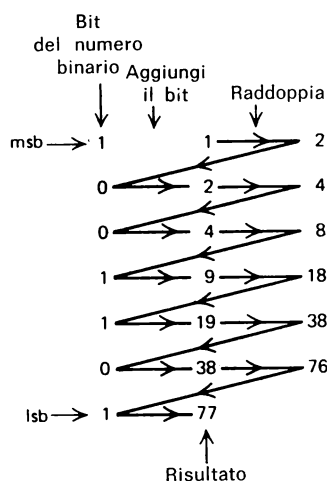



FIG. 5.1. - Conversione da binario a decimale con il metodo *double-dabble*.

cessive divisioni per 2 del numero intero decimale; i resti successivi costituiscono le cifre del numero binario, il primo resto essendo il bit meno significativo.

Esempio 5.5. Per convertire il numero intero 167_{10} in numero binario si può procedere nel modo seguente:

	RESTO	PESO BINARIO
167 : 2 = 82	1	2^0 (lsb)
83 : 2 = 41	1	2^1
41 : 2 = 20	1	2_2
20 : 2 = 10	0	2^3
10 : 2 = 5	0	2^4
5 : 2 = 2	1	2^5
2 : 2 = 1	0	2^6
1 : 2 = 0	1	2^7 (msb)

La risposta si ottiene leggendo la colonna dei resti dal basso verso l'alto: $167_{10} = 10100111_2$.

Quando il numero è frazionario, esso viene sottoposto a successive moltiplicazioni per 2, e il risultato si ottiene leggendo la colonna dei riporti dall'alto verso il basso. Talvolta un numero frazionario in base dieci non può essere convertito in un numero frazionario in base due composto con un numero finito di bit; in ta caso il processo di conversione deve essere troncato, dando luogo ad un errore di *arrotondamento* (*round-off*).

Esempio 5.6. Per convertire il numero 0.57_{10} in un numero binario, si procede come segue:

	RIPORTO	PESO BINARIO
$0.57 \times 2 = 1.14$	1	2^{-1} (msb)
$0.14 \times 2 = 0.28$	0	2^{-2}
$0.28 \times 2 = 0.56$	0	2^{-3}
$0.56 \times 2 = 1.12$	1	2^{-4}
$0.12 \times 2 = 0.24$	0	2^{-5}
$0.24 \times 2 = 0.48$	0	2^{-6}
$0.48 \times 2 = 0.96$	0	2^{-7}
$0.96 \times 2 = 1.92$	1	2^{-8}

A questo punto si decide di troncare il processo di conversione. Il risultato si ha leggendo la colonna dei riporti dall'alto verso il basso; e risulta $0.57_{10} = 0.10010001 + \varepsilon$, con un errore di troncamento $\varepsilon < 2^{-8} = 1/256$.

La conversione di un numero misto costituito da una parte intera e da una parte frazionaria può essere effettuata nel modo seguente: si separano parte intera e frazionaria, si convertono queste parti separatamente mediante i rispettivi algoritmi di conversione, e si fondono infine i due risultati in maniera appropriata.

Rappresentazione dei numeri negativi

Finora si sono considerati numeri interi e frazionari binari senza riguardo al segno; si descrivono ora tre rappresentazioni mediante le quali si possono distinguere numeri positivi e negativi. Ciascuna di queste tre rappresentazioni impiega un bit addizionale, il *bit del segno* (*sign bit*), che diventa il bit più a sinistra nei *numeri binari con segno*. Le tre rappresentazioni sono la *rappresentazione del segno e del modulo*, la *rappresentazione con complemento a 1*, e la *rappresentazione con complemento a 2.*

RAPPRESENTAZIONE IN SEGNO E MODULO. In questa rappresentazione il modulo del numero è espresso in forma binaria (equazione 5.1); il bit del segno è posto alla sinistra del bit più significativo del modulo; se è « 0 » rappresenta un « + » e se è « 1 » rappresenta un « – ». Di modo che il numero positivo $+ 5.25_{10}$ è rappresentato da 0101.01, mentre il numero negativo $- 5.25_{10}$ è rappresentato da 1101.01. Un numero binario di n cifre intere e m cifre frazionarie nella rappresentazione in segno e modulo è espresso da $n + m + 1$ bit e vale:

$$N = (-1)^{b_n} \cdot \left(\sum_{i=1}^{n-1} b_i 2^i + \sum_{i=-m}^{-1} b_i 2^i \right)$$

$$= (-1)^{b_n} \cdot \sum_{i=-m}^{n-1} b_i 2^i, \qquad (5.5)$$

dove b_i sono i coefficienti binari 0 o 1.

Questa rappresentazione, sebbene facilmente comprensibile, non si presta ad una facile realizzazione dei circuiti aritmetici digitali per diverse ragioni. Per esempio, il numero zero viene ad avere due differenti espressioni: 000...0 e 100...0; inoltre, quando si sommano un numero positivo e uno negativo, prima si deve ricercare il numero di modulo maggiore e poi sottrarre a questo quello di modulo minore, e infine assegnare al risultato il segno del numero che ha modulo maggiore.

Rappresentazione con complemento ad 1. Nella rappresentazione con complemento ad 1, i numeri positivi sono espressi esattamente nello stesso modo che nella rappresentazione in segno e modulo; medesima in entrambe le rappresentazioni è anche la convenzione per il bit del segno e cioè 0 per « + » e 1 per « − ». Cambia invece il modo di esprimere il modulo di un numero negativo, che viene rappresentato con il suo *complemento ad 1.* Il complemento ad 1 di un numero binario si ottiene mediante *inversione logica* di ciascun bit del numero, vale a dire cambiando ogni 0 in 1 e ogni 1 in 0. Si noti che il complemento del complemento di un numero è il numero stesso; si noti pure che in questa rappresentazione, a differenza che in quella in segno e modulo, il bit del segno, che è il bit più significativo, è trattato nell'addizione e nella sottrazione allo stesso modo dei bit che rappresentano il modulo.

Esempio 5.7. Nella rappresentazione con complemento ad 1, il numero positivo + 5.25 è espresso con $0\,101.01_2$, come nella rappresentazione in segno e modulo. Per esprimere invece in questa rappresentazione il numero $-\,5.25_{10}$, prima si costruisce il complemento del modulo, ossia di 101.01_2 che è 010.10, quindi si pone alla sinistra del bit più significativo il bit del segno meno, che è 1; in conclusione, nella rappresentazione con complemento ad 1 si ha $-\,5.25_{10} =$ $= 1\,010.10_2$.

Si può mostrare che un numero binario di n bit interi e m bit frazionari, espresso nella rappresentazione con complemento ad 1 da $n + m + 1$ bit, vale:

$$N = (1 - b_n) \cdot \sum_{i=-m}^{n-1} b_i 2^i - b_n \cdot \sum_{i=-m}^{n-1} (1 - b_i)\, 2^i \tag{5.6}$$

dove b_i sono i coefficienti binari 0 o 1. Si noti che $1 - b_i$ è il complemento ad 1 del bit i-esimo. Inoltre, per i numeri positivi il secondo termine della parte destra dell'equazione 5.6 si riduce a zero, mentre per i numeri negativi diventa zero il primo termine.

Si noti inoltre che, come già per il caso della rappresentazione in segno e modulo, nella rappresentazione con complemento ad 1 il numero 0 presenta due differenti espressioni, che ora, però, sono 00...0 e 11...1.

RAPPRESENTAZIONE CON COMPLEMENTO A 2. La rappresentazione dei numeri con complemento a 2 è largamente adoperata nei microcomputer. I numeri positivi sono espressi esattamente nello stesso modo nella rappresentazione in segno e modulo, in quella con complemento ad 1 e in quella con complemento a 2; inoltre, anche il significato del bit del segno è il medesimo nelle tre rappresentazioni: 0 per « + » e 1 per « − ». Nella rappresentazione con complemento a 2, per esprimere il modulo di un numero, si prende appunto il suo *complemento a 2.* Il complemento a 2 di un numero si può ottenere ricavando prima il suo complemento ad 1 e aggiungendo poi 1 al bit meno significativo, intero o frazionario che sia. Si noti che, come già nel caso della rappresentazione con complemento ad 1, nell'addizione e nella sottrazione il bit del segno è trattato allo stesso modo dei bit che rappresentano il modulo.

Esempio 5.8. (*a*) Il complemento a 2 di $58_{10} = 111010_2$ si ricava come segue:

$$\begin{array}{lrl} & 58_{10} = & 0\ 111010 \\ & \text{complemento ad 1 di } 58_{10} = & 1\ 000101 \\ & \text{addizione di 1 a lsb}: & 1 \\ \hline \text{risultato:} & \text{complemento a 2 di } 58_{10} = & 1\ 000110 \end{array} \quad \left.\right\} \text{addiz.}$$

(*b*) Il complemento a 2 di 42.5_{10} si ottiene come segue:

$$\begin{array}{lrl} & 42.5_{10} = & 0\ 101010.1 \\ & \text{complemento ad 1 di } 42.5_{10} = & 1\ 010101.0 \\ & \text{addizione di 1 a lsb}: & 1 \\ \hline \text{risultato:} & \text{complemento a 2 di } 42.5_{10} = & 1\ 010101.1 \end{array} \quad \left.\right\} \text{addiz.}$$

Si può mostrare che un numero binario costituito da n bit interi e m bit frazionari, espresso nella rappresentazione con complemento a 2 da $n + m + 1$ bit, ha il valore:

$$N = (1 - b_n) \cdot \sum_{i=-m}^{n-1} b_i 2^i - b_n \cdot \sum_{i=-m}^{n-1} [(1 - b_i)2^i] + 2^{-m} \tag{5.7}$$

Per i numeri interi l'equazione 5.7 si riduce a:

$$N = (1 - b_n) \cdot \sum_{i=0}^{n-1} b_i 2^i - b_n \cdot \sum_{i=0}^{n-1} [(1 - b_i)2^i] + 1 \tag{5.8}$$

Si noti che, a differenza delle altre due rappresentazioni, la rappresentazione con complemento a 2 possiede una sola espressione per il numero 0, e cioè 000...0. La Tav. 5.1 mostra alcuni numeri espressi nelle tre rappresentazioni discusse precedentemente.

TABELLA 5.1

Tre rappresentazioni binarie di alcuni numeri

Numero decimale	Rappresentazione in segno e modulo	Rappresentazione in complem. ad 1	Rappresentazione in complem. a 2
+ 7	0 111	0 111	0 111
+ 6	0 110	0 110	0 110
+ 5	0 101	0 101	0 101
+ 4	0 100	0 100	0 100
+ 3	0 011	0 011	0 011
+ 2	0 010	0 010	0 010
+ 1	0 001	0 001	0 001
0	{0 000 1 000}	{0 000 1 111}	0 000
− 1	1 001	1 110	1 111
− 2	1 010	1 101	1 110
− 3	1 011	1 100	1 101
− 4	1 100	1 011	1 100
− 5	1 101	1 010	1 011
− 6	1 110	1 001	1 010
− 7	1 111	1 000	1 001

5.2. Rappresentazione dei numeri in ottale e in esadecimale

Un operatore umano non manipola agevolmente le lunghe strisce di 1 e di 0 utilizzate per rappresentare i numeri. Due sono le soluzioni generalmente adottate per superare questo problema: la *codifica* (*coding* o *encoding*), che è discussa nel successivo paragrafo, e l'uso di numeri espressi in una base costituita da una potenza intera di 2, come i *numeri ottali* (base (2^3) e i numeri *esadecimali* (base 2^4).

Sistema di numerazione ottale

In questo sistema, la base è $b = 2^3 = 8_{10}$; per esprimere i numeri sono quindi necessari otto simboli, dallo 0 al 7. Un numero ottale N_8 costituito da n cifre intere e m cifre frazionarie assume il valore della somma:

$$N_8 = a_{n-1}\,8^{n-1} + a_{n-2}\,8^{n-2} + \ldots + a_1 8^1 + a_0 8^0 + a_{-1}\,8^{-1} + \ldots + \\ + a_{-m}\,8^{-m} = \sum_{i=-m}^{n-1} a_i 8^i \qquad (5.9)$$

Gli otto simboli necessari nel sistema di numerazione ottale possono essere rappresentati dalle combinazioni di 3 bit binari. La conversione di un numero binario in un numero ottale può quindi essere ottenuta con la regola seguente: si suddivide la parte intera del numero binario in gruppi di 3 cifre, partendo dalla cifra intera più a destra; la parte frazionaria è suddivisa in modo simile, partendo però dal bit frazionario più a sinistra.

Esempio 5.9. Per convertire il numero binario 11100011.1011 in un numero ottale si fa come segue:

```
           ┌─0 di testa implicita
           ↓                   ↓↓─0 di coda impliciti
Binario:  011  100  011  .  101  100

Ottale:    3    4    3   .   5    4
```

Nella Tab. 5.2 è riportata la rappresentazione in ottale di alcuni numeri.

Sistema di numerazione esadecimale

In questo sistema la base è $b = 2^4$, per cui si richiedono $2^4 = 16_{10}$ simboli. I simboli usati sono 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; A rappresenta il dieci, B l'undici, e così via fino a F, che rappresenta il quindici. Nella Tab. 5.2 è riportata la rappresentazione esadecimale di alcuni numeri.

TABELLA 5.2

Rappresentazione decimale, binaria, ottale, esadecimale

Decimale	Binaria	Ottale	Esadecimale
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

I 16 differenti simboli possono essere rappresentati esattamente con le combinazioni di 4 bit. In tal modo una regola per convertire un numero binario in un numero esadecimale è la seguente: si suddivide la parte intera di un numero binario in gruppi di 4 cifre, partendo dal

bit intero più a destra; la parte frazionaria va suddivisa in modo simile, partendo però dalla cifra frazionaria più a sinistra.

Esempio 5.10. Convertire il numero binario 11100011.1011 (= 227.6875_{10}) in un numero esadecimale. Si ha:

Binario: 1110 0011 . 1011

Esadecimale: E 3 . B

Confronto fra i sistemi numerici

Un numero N nella rappesentazione binaria risulta espresso da b cifre, nella rappresentazione ottale da e cifre, nella rappresentazione decimale da d cifre e nella rappresentazione esadecimale da h cifre, dove N, b, e, d, e h sono legati dalla

$$N = 2^b{}_{10} = 8^e{}_{10} = 10^d{}_{10} = 16^h{}_{10} \tag{5.10}$$

Dall'equazione 5.10 segue allora che:

$$\frac{e}{b} = 3, \quad \frac{h}{b} = 4, \quad \frac{h}{e} = \frac{4}{3}, \quad \frac{b}{d} = \frac{1}{\log_{10} 2} = 3.82,$$

$$\frac{e}{d} = \frac{1}{\log_{10} 8} = 1.05, \quad \frac{h}{d} = \frac{1}{\log_{10} 16} = 0.895 \tag{5.11}$$

La conversione da binario a decimale descritta nell'Esempio 5.5 consisteva nel dividere ripetutamente per 2 il numero decimale intero per ottenere il numero binario equivalente; il procedimento può essere generalizzato nel modo seguente. Se un numero in base b_1 deve essere convertito in un numero in base b_2, si devono effettuare ripetute divisioni per b_2 se il numero è intero e ripetute moltiplicazioni per b_2 se il numero è frazionario; in entrambi i casi le operazioni vanno eseguite nell'aritmetica in base b_1. Così la conversione dall'ottale o dall'esadecimale al decimale richiede divisioni e moltiplicazioni in ottale o esadecimale, rispettivamente.

Tali conversioni e le operazioni aritmetiche in ottale e in esadecimale in genere sono facilitate dall'uso delle tavole di addizione e di moltiplicazione riportate nelle Appendici A e B.

5.3. Codificazione

Quattro sono le applicazioni principali della codificazione in un microcomputer: (1) riduzione del numero delle cifre ad un livello che può essere trattato convenientemente da un operatore umano; (2) codificazione in binario delle cifre dei numeri decimali, conveniente quale rappresentazione I/O; (3) codificazione per rilevare errori originatisi durante la trasmissione di informazioni fra le varie parti di un computer; e (4) codificazione per rappresentare in forma binaria caratteri dell'alfabeto e altri simboli, inclusi quelli richiesti per la comunicazione fra il computer e i suoi dispositivi periferici (per esempio, « ritorno del carrello » di una stampante elettrica).

Codificazione per ridurre il numero delle cifre

I numeri ottali e esadecimali discussi nel paragrafo 5.2 sono buoni esempi di rappresentazioni di numeri che richiedono meno cifre che la rappresentazione binaria.

> **Esempio 5.11.** Il numero di cifre richiesto per esprimere un numero intero in rappresentazione ottale è minore per un fattore 3 di quello richiesto per la rappresentazione binaria, mentre la rappresentazione esadecimale dà luogo a una riduzione per un fattore 4.

Numeri decimali codificati in binario

La rappresentazione decimale codificata in binario (*BCD*: *binary-coded decimal*) utilizza 4 bit binari per ottenere 10 combinazioni per i numeri decimali da 0 a 9. In effetti, 4 bit binari possono rappresentare fino a 16 numeri; si ha quindi una ridondanza che dà luogo a differenti codici BCD. Tre di tali codici sono mostrati nella Tav. 5.3. Il codice BCD usato più comunemente è il 8-4-2-1 o codice dei *decimali codificati in binario con peso naturale*, mostrato nella colonna 2 della tavola. La colonna 3 mostra il codice *BCD* 2-4-2-1, mentre l'ultima colonna mostra il codice *BCD* con eccesso 3, che è derivato dal codice *BCD* 8-4-2-1 aggiungendo 3_{10} a ciascun numero codificato. Gli

TABELLA 5.3

Numeri BCD

Numeri decimali	BCD 8821	BCD 2421	BCD eccesso 3
0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 1 0 0	1 0 0 1
7	0 1 1 1	1 1 0 1	1 0 1 0
8	1 0 0 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 0 0

ultimi due codici hanno proprietà di simmetria, illustrate dalla linea tratteggiata, che facilitano la *rappresentazione con complemento a 9* utile nella sottrazione BCD.*

Codici a rivelazione e a correzione di errore

Sono stati sviluppati numerosi codici che rivelano e anche correggono gli errori che si producono nei dati durante il loro transito lungo i vari percorsi entro il computer e fra il computer e i suoi dispositivi periferici. Tutti questi codici richiedono dei bit di *controllo* (*check bit*) supplementari per rivelare la presenza degli errori e per indicare il bit o i bit errati. Prendiamo qui in esame solamente i codici a rivelazione di errore più semplici e più largamente utilizzati, vale a dire i codici a controllo di *parità* (*even parity*) o di *disparità* (*odd parity*). Tali codici migliorano l'affidabilità della trasmissione

* Una discussione dettagliata sulla codificazione può essere trovata in Bibliografia 1.

con l'aggiunta di un bit a quelli relativi all'informazione codificata. Nel caso del controllo di disparità, il bit di controllo supplementare, p, ha un valore tale che:

$$p \oplus X_n \oplus X_{n-1} \oplus \ldots \oplus X_1 \oplus X_0 = 1, \tag{5.12}$$

dove il simbolo $\oplus$ indica « somma modulo-2 »: $1 \oplus 0 = 0 \oplus 1 = 1$, $1 \oplus 1 = 0$, $1 \oplus 1 \oplus 1 = 1$, e così via. Nel caso del controllo di parità, il bit di check p è scelto in modo che:

$$p \oplus X_n \oplus X_{n-1} \oplus \ldots \oplus X_1 \oplus X_0 = 0 \tag{5.13}$$

Esempio 5.12. Nella Tab. 5.4 sono mostrati i bit dal controllo di parità e quelli del controllo di disparità relativi ai primi 10 numeri in BCD.

TABELLA 5.4

Controllo di parità e di disparità

Numeri decimali	BCD 8421	Bit di controllo	
		Disparità	Parità
0	0 0 0 0	1	0
1	0 0 0 1	0	1
2	0 0 1 0	0	1
3	0 0 1 1	1	0
4	0 1 0 0	0	1
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	0	1
8	1 0 0 0	0	1
9	1 0 0 1	1	0

Si noti che sia il controllo di parità sia quello di disparità rivelano solamente la presenza di un numero dispari di errori; va detto però che la probabilità di errori doppi, quadrupli, e così via è trascurabile quando il tasso di errore singolo è basso. Ciò nonostante, si

sono sviluppati anche codici molto sofisticati a rivelazione di errori multipli; tali codici non vengono qui considerati.

Codificazione di caratteri alfabetici e di altri simboli (Codici ASCII)

Sono stati sviluppati diversi codici per rappresentare in forma binaria i caratteri dell'alfabeto e gli altri simboli di solito presenti sulla tastiera di una telescrivente. Uno di tali codici, che ha ottenuto una larga diffusione nei sistemi a microcomputer, è il *codice ASCII*; ASCII è un acronimo per *American Standard Code for Information Interchange.*

Il codice ASCII completo impiega 8 bit, per cui può rappresentare 256 caratteri e simboli, comprese le lettere maiuscole e le lettere minuscole. Poiché numerosi microcomputer hanno parole con lunghezza multipla di 4 bit, il codice ASCII prevalente è il codice a 8 bit; tuttavia, sono stati sviluppati anche codici ASCII ridotti, a 6 bit e a 7 bit, che possono essere usati a spese di uno scomodo procedimento d'*impacchettamento*, vale a dire ricorrendo a pesante programmazione.

5.4. Rappresentazione e aritmetica in virgola mobile

Rappresentazione in virgola mobile

La rappresentazione dei numeri in *virgola mobile* (*floating-point*), simile alla notazione scientifica, è importante per conservare la massima precisione quando si opera con numeri molto grandi o molto piccoli. Essa fa intervenire due quantità: l'*esponente E* che vien detto anche la *caratteristica*, e la *mantissa M* che viene anche indicata come *campo frazionario.*

> **Esempio 5.13.** La velocità della luce, che è 300 000 000 metri al secondo, può essere espressa in notazione scientifica come 0.3×10^9 metri/secondo, e nella rappresentazione in virgola mobile con l'esponente 9 e la mantissa 0.3.

Il numero di parole richieste per esprimere E e M è stabilito dalla precisione desiderata e dalla lunghezza della parola del micro-

computer. La mantissa M è scelta in modo che:

$$1/2 \leq |M| < 1 \tag{5.14}$$

Sia l'esponente che la mantissa possono essere espressi nella rappresentazione con complemento a 2, e il valore della mantissa M deve essere *normalizzato* in modo da soddisfare all'equazione 5.14.

Esempio 5.14. Per esprimere $0.0074_8 = 0.000000111100_2$ nella rappresentazione in virgola mobile, si deve normalizzare M perché non soddisfa l'equazione 5.14. La normalizzazione viene ottenuta portando il numero binario entro l'intervallo dato dall'equazione 5.14, ossia muovendo il punto binario di 6 posti a destra. Quindi il numero 0.0074_8 nella rappresentazione in virgola mobile è espresso dall'esponente $E = 1\ 1010$ e dalla mantissa $M = 0\ 1111$, dove E è rappresentato in complemento a 2 e in M sono stati omessi gli zeri in coda.

In fig. 5.2 è mostrato un numero rappresentato in virgola mobile mediante quattro parole di 8 bit. Il bit più significativo della prima parola è utilizzato come bit del segno della mantissa; i restanti 7 bit esprimono l'esponente in notazione *eccesso 64*, nella quale cioè 64_{10} è sommato all'esponente per poter accogliere esponenti positivi e negativi senza usare il bit del segno; la seconda, la terza e la quarta parola costituiscono i 24 bit della mantissa. Si può mostrare che l'intervallo N di numeri rappresentati delle quattro parole di fig. 5.2 risulta (cfr. Problema 10):

$$2^{-65} \leq N \leq (1 - 2^{-24}) \cdot 2^{63} \tag{5.15}$$

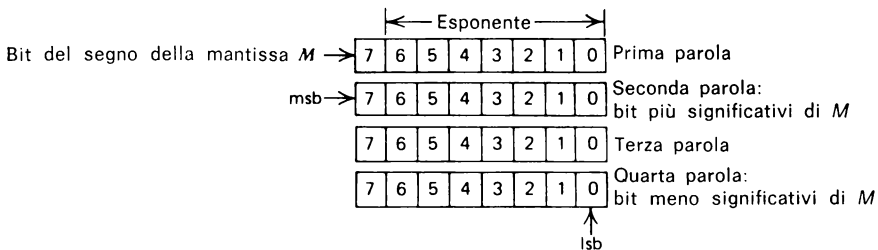



FIG. 5.2. - Rappresentazione di un numero in virgola mobile mediante quattro parole di 8 bit.

Aritmetica in virgola mobile

Nella somma o nella sottrazione di due numeri espressi in virgola mobile, i due numeri vanno prima *allineati*, in modo che l'esponente del numero più grande diventi l'esponente del risultato.

Nella moltiplicazione, gli esponenti sono sommati mentre le mantisse sono moltiplicate:

$$X \cdot Y = (2^{E_X + E_Y}) \cdot (M_X \cdot M_Y) \tag{5.16}$$

nella divisione, gli esponenti vanno sottratti e le mantisse divise:

$$X : Y = (2^{E_X - E_Y}) \cdot (M_X : M_Y) \tag{5.17}$$

Il risultato di un'operazione aritmetica eseguita su numeri in virgola mobile richiede anche la normalizzazione se la mantissa del risultato è esterna all'intervallo indicato nell'equazione 5.14.

Problemi

1. Convertire in decimale i numeri binari seguenti: 101101, 0.101110, 110.101.
2. Determinare l'equivalente binario dei seguenti numeri decimali: 23, 0.28125, 224.375.
3. Scrivere l'equazione 5.3 in una forma generale che sia applicabile ad ogni base *b*.
4. Confrontare le equazioni 5.1 e 5.4 e stabilire la validità dell'algoritmo dell'esempio 5.3 (*double-dabble*) per la conversione da binario a decimale.
5. Sono dati i seguenti numeri binari: 1 1010.01, 0 1010.01, 1 110010, 0 110010, 1 0100.11, 0 0100.11. Usare le equazioni da 5.5 a 5.8 per trovare i loro equivalenti decimali se i numeri sono (*a*) nella rappresentazione in segno e modulo, (*b*) nella rappresentazione in complemento ad 1, e (*c*) nella rappresentazione in complemento a 2.
6. Esprimere i numeri seguenti nella rappresentazione binaria in segno e modulo, nella rappresentazione in complemento ad 1,

nella rappresentazione in complemento a 2: -1001.5_{10} -853_{10}, -0.153_{10}, -52.0625_{10}. L'errore di arrotondamento deve essere più piccolo della cifra meno significativa fra quelle date.

7. Esprimere i numeri seguenti in segno e modulo nel codice BCD 8-4-2-1: -93.2_{10}, -107.25_{10}, -0.769_{10}.

8. Stabilire gli algoritmi per l'addizione e per la sottrazione nel codice BCD con eccesso 3. Considerare le correzioni da apportare quando: (*a*) il risultato di un'addizione è minore di 10_{10} nel codice considerato; (*b*) per effetto dell'addizione si produce un riporto alla cifra BCD immediatamente successiva; (*c*) il risultato di una sottrazione è minore di 10_{10}; (*d*) in una sottrazione si genera un prestito dalla cifra BCD immediatamente successiva.

9. Esprimere i numeri seguenti nella rappresentazione in virgola mobile: $+364_8$, -27.3_8, $+0A.04_{16}$, -0.025_{16}.

10. Verificare l'equazione 5.15. *Suggerimento*: il più piccolo valore di $|M|$ è 2^{-1} e il più grande è $(2^{24}-1)\cdot 2^{-24}$.

11. Discutere lo spostamento che si rende necessario per effettuare l'addizione di due numeri nella rappresentazione in virgola mobile quando un numero ha un esponente positivo e l'altro numero un esponente negativo.

12. Usare la rappresentazione binaria in virgola mobile e calcolare $763.2_{10} - 0.421_{10}$.

6. CIRCUITI ARITMETICI E LOGICI

In un microcomputer le operazioni aritmetiche e quelle logiche sono eseguite dai *circuiti aritmetici e logici.* Fra questi circuiti, quelli per le operazioni logiche e quelli per l'addizione e la sottrazione sono normalmente combinati in una *unità aritmetico-logica* (*ALU*: *arithmetic-logic unit*). Questo capitolo fornisce alcuni dettagli circa gli *addizionatori* e i *sottrattori binari*, gli *addizionatori binario-decimali* e le funzioni logiche della ALU, ed inoltre presenta una breve descrizione dell'accumulatore e dei *circuiti moltiplicatori* e *divisori.*

6.1. Addizionatori e sottrattori

Addizionatori binari

In un microcomputer, l'addizione di due numeri viene eseguita prelevando il primo operando dall'accumulatore (registro A), il secondo operando da un altro registro o dalla memoria principale, e addizionandoli poi in un circuito addizionatore; il risultato viene posto nell'accumulatore, dove cancella il contenuto precedente. In questo modo si possono eseguire anche addizioni successive; dopo ogni operazone l'accumulatore accoglie la nuova somma. Un addizionatore comprende anche un *indicatore di superamento di capacità* (*overflow flag*) e/o un *indicatore di riporto* (*carry flag*), che in sostanza è un bit supplementare dell'accumulatore. Quando un flag segnala il superamento o il riporto, la CPU si limita a riconoscere il fatto, senza dar luogo ad alcuna azione automatica; spetta al programmatore esaminare

lo stato del flag, agire secondo la sua indicazione e cancellare il flag per l'uso successivo.

Il circuito di base per l'addizione è l'*addizionatore completo* (*full adder*) mostrato in fig. 6.1 *a*, nella quale A_i e B_i sono i bit rispettivamente del primo e del secondo operando, S_i è la somma, C_{in} è l'ingres-

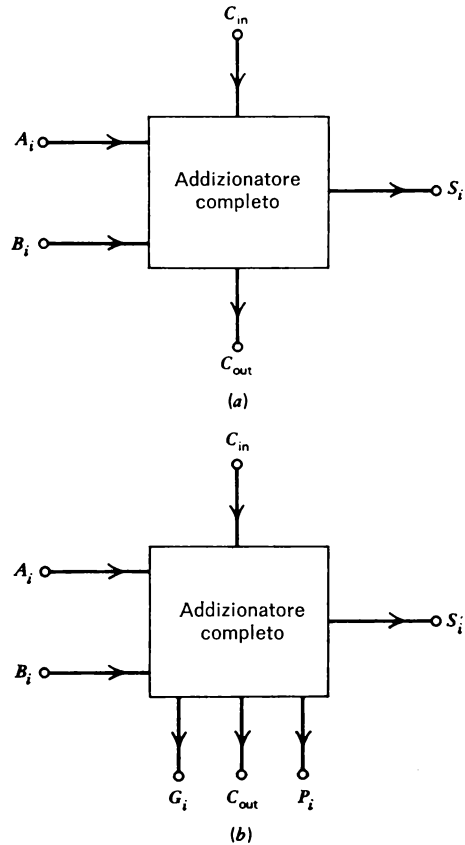



FIG. 6.1. - Addizionatore completo: (*a*) con ingresso del riporto C_{in} e uscita del riporto C_{out}; (*b*) con ingresso del riporto C_{in}, uscita del riporto C_{out} e uscite del riporto *look-ahead* G_i e P_i.

so del riporto e C_{out} è l'uscita del riporto. Questo addizonatore completo è in grado di effettuare l'addizione *seriale* (ossia bit per bit), ed inoltre costituisce il blocco componente di base per gli *addizionatori in parallelo*.

Il più semplice addizionatore parallelo è l'addizionatore *ad onda* (*ripple adder*) che è realizzato mettendo in cascata *n* addizionatori completi per l'addizione di due numeri di *n* bit. Con questa soluzione

il segnale di riporto, C_{out}, si deve propagare attraverso n stadi successivi, il che può provocare un ritardo importante nell'ottenimento del risultato finale. Si può ottenere un addizionatore più veloce a spese però di una circuiteria più complicata. Nell'addizionatore a riporto simultaneo (*look ahead carry adder* o *simultaneous-carry adder*) mostrato in fig. 6.1 *b*, per ogni bit vengono generati due segnali ausiliari G_i (*generate*) e P_i (*propagate*), dove $G_i = A_i B_i$ e $P_i = A_i \oplus B_i = A_i \bar{B}_i + \bar{A}_i B_i$ *. Si può mostrare (Bibliografia 1) che gli ingressi e le uscite del riporto in un addizionatore di questo tipo per 4 bit sono dati dalle equazioni:

$$C_{in_0} = 0, \tag{6.1a}$$

$$C_{in_1} = G_0, \tag{6.1b}$$

$$C_{in_2} = G_1 + G_0P_1, \tag{6.1c}$$

$$C_{in_3} = G_2 + G_1P_2 + G_0P_1P_2, \tag{6.1d}$$

$$C_{out} = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3. \tag{6.1e}$$

Si noti che il ritardo di propagazione dell'uscita del riporto C_{out} in un addizionatore a riporto simultaneo è minore che in un addizionatore ad onda, perché C_{out} non deve propagarsi attraverso 4 sommatori completi ma soltanto attraverso due livelli di circuiteria logica. In fig. 6.2 è indicato un addizionatore a riporto simultaneo per 4 bit. Disponendo in cascata n di tali addizionatori si può ottenere un *addizionatore ibrido* per 4 *n bit*.

Sottrattori binari

Perché i numeri negativi possono essere espressi in tre diverse rappresentazioni, ci sono almeno tre tipi di sottrattori: sottrattori in *segno e modulo*, sottrattori in *complemento ad 1* e sottrattori in *complemento a 2*.

SOTTRATTORI IN SEGNO E MODULO. Il blocco funzionale fondamentale è il *sottrattore completo* (*full subtractor*), simile all'addizionatore

* In questo capitolo il segno « + » indica l'operazione logica OR.

completo; analogamente agli addizionatori a riporto simultaneo, si possono anche costruire sottrattori a *prestito simultaneo* (*look-ahead borrow*). Quando si esegue la sottrazione $A - B$, si deve prima stabilire quale dei due numeri A e B ha modulo maggiore; il risultato avrà

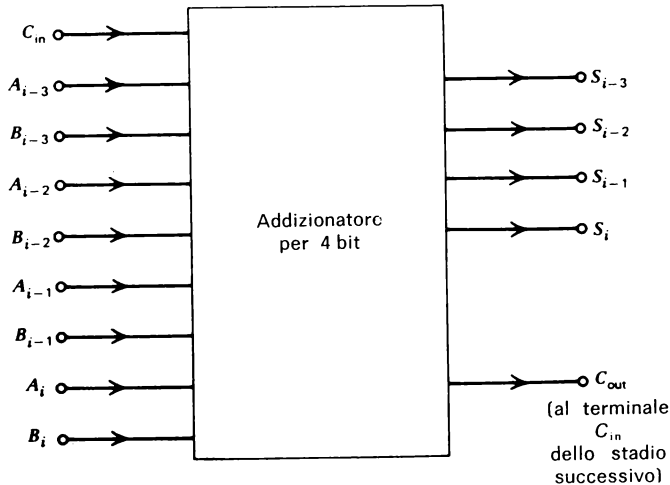



FIG. 6.2. - Addizionatore *look-ahead* per 4 bit costituito da quattro circuiti di fig. 6.1 *b* e da circuiti logici che forniscono i riporti secondo le equazioni 6.1.

il segno di questo numero. Il modulo del risultato si ottiene con $A - B$ se $A > B$ e con $B - A$ se $A < B$.

SOTTRATTORI IN COMPLEMENTO AD 1. In questa rappresentazione la differenza $A - B$ può essere ottenuta come $-(B - A)$ usando un addizionatore completo e addizionando a B il complemento ad 1 di A. In questo circuito però è necessaria una correzione quando $A - B$ è negativo; tale correzione può essere ottenuta ricorrendo ad un riporto *end around* da C_{out} a C_{in} come mostrato in fig. 6.3.

Esempio 6.1. La differenza $A - B$ con $A = 35_8$ *e* $B = 57_8$ può essere ottenuta con il circuito di fig. 6.3 secondo lo schema seguente:

$B =$	$57_8 =$	0 101111	
$-A =$	$-35_8 =$	1 100010	← complemento ad 1 di A
		1|0 010001	← somma scorretta
		1	← riporto *end-around*
$B - A$	$=$	0 010010	← somma del riporto *end-around* e della somma scorretta
$A - B$	$=$	1 101101	← rappresentazione di $(A - B)$ in *complemento ad 1*

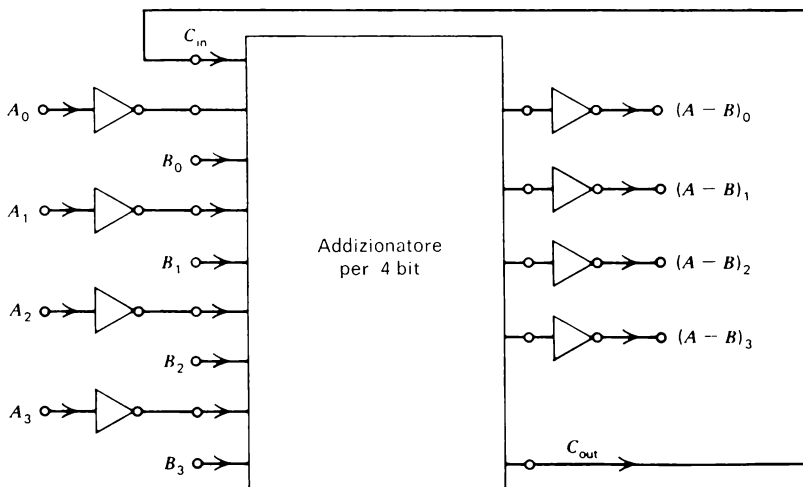



FIG. 6.3. - Sottrattore in complemento ad 1 che impiega l'addizionatore per 4 bit di fig. 6.2 e un riporto *end-around.*

SOTTRATTORI IN COMPLEMENTO A 2. In questa rappresentazione, la differenza $A - B$ è ottenuta ricavando dapprima il complemento a 2 del termine B e sommandolo poi al termine A. Le operazioni da fare sono le seguenti, nell'ordine: inversione di tutti i bit di B mediante l'uso di inveritori; addizione di 1 al risultato per ottenere il complemento a 2 di B; addizione di A al risultato precedente.

Addizionatori binario-decimali

In alcuni microcomputer, specialmente in quelli dedicati alle operazioni aritmetiche (*calculator*), queste operazioni sono eseguite in un

sistema binario-decimale, per lo più in BCD 8-4-2-1. Mentre le operazioni sui singoli bit di una cifra decimale codificata in binario sono eseguite in parallelo, quelle sulle singole cifre invece sono spesso eseguite serialmente; l'addizionatore risulta in tal caso un circuito per 4 bit.

La fig. 6.4 illustra l'addizione di due numeri binario-decimali, contenuti rispettivamente nell'accumulatore (registro A) e nel registro B. Le cifre meno significative vengono addizionate per prime, dando origine alla cifra meno significativa della somma e ad un eventuale riporto. In corrispondenza del successivo impulso d'orologio (*clock*), il contenuto dell'accumulatore e quello del registro B sono spostati a destra di una cifra decimale, la somma ottenuta viene introdotta nel-

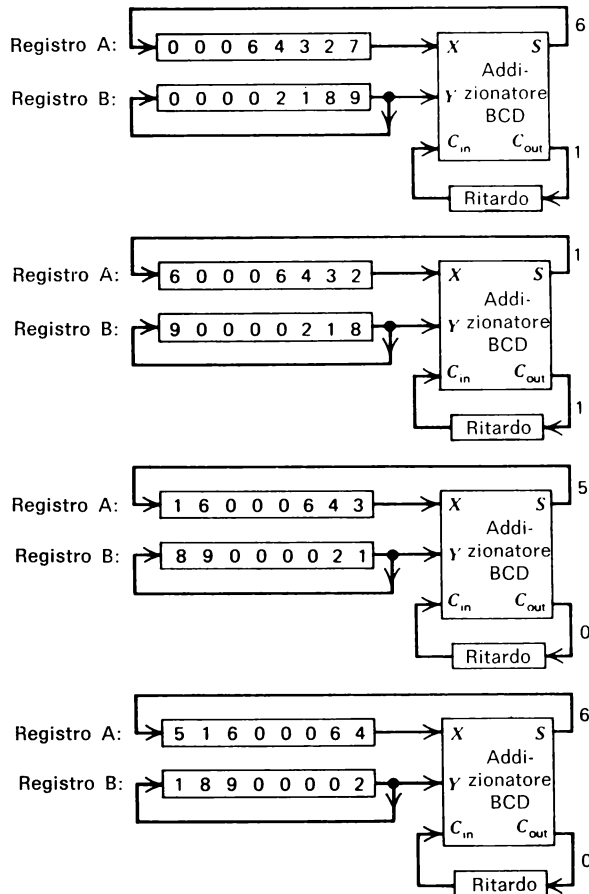



FIG. 6.4. - Tre fasi dell'addizione in BCD dei numeri 64327_{10} e 2189_{10}. Ciascun addizionatore BCD contiene un addizionatore completo binario per 4 bit e un circuito di correzione.

l'accumulatore, mentre il riporto entra in un circuito che lo ritarda per un periodo di clock, per essere sommato con le cifre del posto successivo. Le operazioni vengono ripetute su tutte le cifre dei numeri A e B (fig. 6.4).

Nel caso dell'addizionatore binario-decimale nel codice 8-4-2-1, se S è la somma di due cifre decimali di ugual posto, si può avere: (1) $0000 \leq S \leq 1001$, nel qual caso il risultato è corretto; (2) $1010 \leq S \leq$ ≤ 1111, nel qual caso il risultato è una combinazione BCD non valida; (3) $10000 \leq S \leq 11001$, nel qual caso si genera un riporto e il risultato, se letto nel codice BCD stabilito, non è corretto. Di conseguenza, l'addizionatore considerato deve comprendere un circuito combinatorio che esamina la somma S e ad essa aggiunge 6_{10} se si verificano le condizioni (2) o (3).

Esempio 6.2$_2$ Questo esempio considera tre casi di addizione binario-decimale, le condizioni che richiedono correzioni, e le correzioni applicate.

(1) $5_{10} = 0101$
$3_{10} = 0011$

---

somma $= 1000 = 8_{10}$ (risultato corretto)

(2) $6_{10} = 0110$
$8_{10} = 1000$

(somma =) 1110 (numero BCD non valido)
0110 (si somma 6_{10})

---

somma $= 1\ 0100 = 14_{10}$ (risultato BCD corretto)

(3) $9_{10} = 1001$
$8_{10} = 1000$

(somma =) 1 0001 (si genera riporto e il risultato non è corretto)
0100 (si somma 6_{10})

---

somma $= 1\ 0111 = 17_{10}$ (risultato BCD corretto)

6.2. Moltiplicatori e divisori

Normalmente nella CPU di un microcomputer non sono compresi circuiti per moltiplicare e per dividere; a questo scopo, spesso i costruttori forniscono sottoprogrammi di biblioteca, fondati su metodi analoghi a quello d'esecuzione manuale.

Sebbene i sottoprogrammi siano adeguati per numerose applicazioni, alcune applicazioni di controllo in tempo reale richiedono tecniche più rapide. Una di queste tecniche fa ricorso a un moltiplicatore combinatorio 4×2, costruito su un unico *chip* semiconduttore, che genera prodotti parziali. Per parole di maggior lunghezza, vengono connesse in modo appropriato alcune di queste unità. Il tempo totale di moltiplicazione è pari al massimo ritardo nella propagazione dei prodotti parziali: in taluni circuiti la moltiplicazione di due numeri di 16 bit ha luogo in circa 600 nanosecondi.

Una tecnica un poco più lenta utilizza come moltiplicatori delle ROM, nella funzione di *tavole di consultazione* (*look-up table*), analoghe alla classica tavola pitagorica.

> **Esempio 6.3.** Un moltiplicatore a ROM per parole di 4 bit ha in uscita $(2^4)^2 = 256$ combinazioni di 8 bit; il numero di bit richiesti alla ROM è quindi $256 \times 8 = 2048$.

La moltiplicazione di due parole di 8 bit, tuttavia, già richiederebbe una ROM di $(2^8)^2 \times 16 \approx 1$ milione di bit, che attualmente non esiste. La situazione risulta però notevolmente semplificata se si impiegano tecniche più sofisticate e si usano alcuni sommatori; in tal modo si riduce considerevolmente il numero di bit richiesti alla ROM (Bibliografia 1).

Anche per i divisori, per parole di breve lunghezza, sono possibili le soluzioni a circuiti combinatori e a ROM.

6.3. L'accumulatore e l'unità aritmetico-logica

L'accumulatore (registro A) occupa un posto importante nella CPU, perché tutti i dati sui quali questa opera devono passare attra-

verso di esso. È realizzato spesso con un registro a scorrimento (*shift register*) con un numero di bit pari alla lunghezza della parola del microcomputer, e con possibilità di entrata e di uscita in parallelo (Bibliografia 1).

Il contenuto dell'accumulatore può essere ruotato o fatto scorrere a destra o sinistra, e queste operazioni possono comprendere anche il bit del riporto, se così è specificato. Le varie possibilità sono illustrate in fig. 6.5.

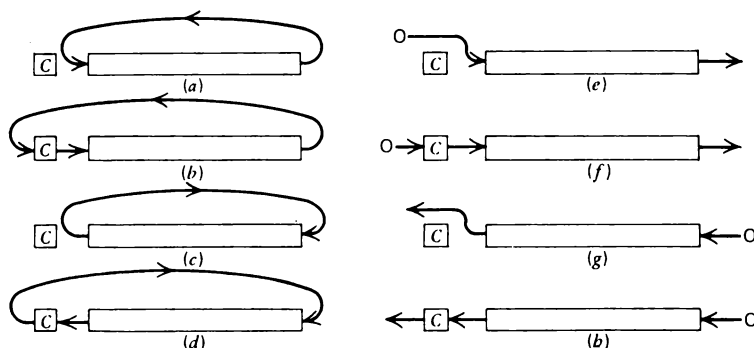



FIG. 6.5. - Operazioni di rotazione e di scorrimento per l'accumualtore: (*a*) rotazione a destra senza riporto; (*b*) rotazione a destra con riporto; (*c*) rotazione a sinistra senza riporto; (*d*) rotazione a sinistra con riporto; (*e*) scorrimento a destra senza riporto; (*f*) scorrimento a destra con riporto; (*g*) scorrimento a sinistra senza riporto; (*h*) scorrimento a sinistra con riporto.

Oltre alle operazioni di rotazione e di scalamento mostrate in fig. 6.5, un microcomputer può anche effettuare le operazioni di *rotazione aritmetica* e di *scalamento aritmetico*. Queste operazioni sono simili alle operazioni in fig. 6.5, salvo che interessano solamente i bit del modulo.

La struttura e le funzioni dell'unità aritmetico-logica (*ALU*: *arithmetic-logic unit*) variano con il microcomputer. In fig. 6.6 è riportata una ALU per 4 bit. Essa è simile ad un addizionatore completo per 4 bit a riporto simultaneo e incorpora circuiti che le consentono di eseguire operazioni aritmetiche e logiche, come mostrato nella Tav. 6.1. L'ingresso MODE stabilisce il tipo di operazione da eseguire, se logica oppure aritmetica, mentre gli ingressi S_2, S_1 e S_0 (*function select*) determinano la funzione specifica, logica o aritmetica.

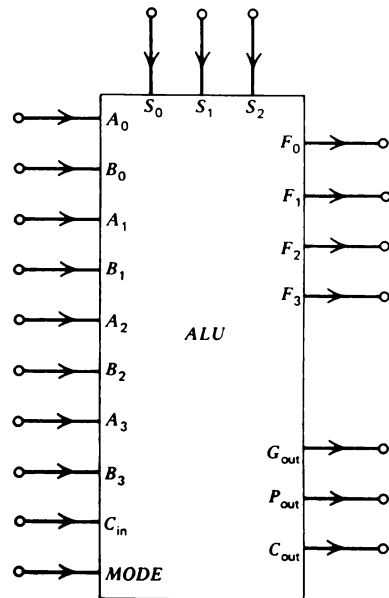



FIG. 6.6. - Schema di una ALU per 4 bit.

TABELLA 6.1

Tabella funzionale di una ALU a 4 bit

Ingressi *function select* $S_2S_1S_0$	$MODE = 1$ operazioni logiche F	$MODE = 0$ operazioni aritmetiche F
0 0 0	$A + B$	A più B
0 0 1	$A \cdot B$	meno 1 (complemento a 2)
0 1 0	$A \oplus B$	A meno 1
0 1 1	$A \odot B$	A più $(A + B)$
1 0 0	0	A per 2
1 0 1	1	A più $(A \cdot B)$
1 1 0	B	A meno B meno 1
1 1 1	A	A

NOTA: I simboli « + », « · », « ⊕ » e « ⊙ » indicano le operazioni logiche *OR*, *AND*, *OR ESCLUSIVO* e *COINCIDENZA*, rispettivamente: « più », « meno » e « per » indicano operazioni aritmetiche.

Le operazioni logiche vengono eseguite bit per bit e comprendono le operazioni AND e OR, l'operazione OR ESCLUSIVO $A \oplus B = A\overline{B} + \overline{A}B$, e l'operazione COINCIDENZA $A \odot B = AB + \overline{A}\overline{B}$. Le operazioni aritmetiche comprendono la somma, il cambiamento di segno, la moltiplicazione per 2 e combinazioni dell'operazione di somma con le operazioni logiche AND e OR.

Per parole di lunghezza superiore a 4 bit si possono mettere in coscata più unità ALU del tipo di fig. 6.6 (*bit slicing*). In tali applicazioni, la propagazione del riporto è facilitata dalle connessioni C_{in}, C_{out}, G_{out}, e P_{out}.

Problemi

1. Usare la rappresentazione con complemento a 2 e fornire esempi di addizione con *overflow* aritmetico.
2. Usare la rappresentazione con complemento a 2 e fornire esempi di addizione nei quali si accende il flag del riporto.
3. Utilizzare l'equazione 6.1 e tracciare lo schema logico di un addizionatore per 4 bit a riporto simultaneo.
4. Utilizzare l'addizionatore per 4 bit di fig. 6.2 e tracciare lo schema blocchi di un addizionatore ibrido per 16 bit. Discutere il ritardo complessivo di propagazione.
5. Completare l'addizione BCD di fig. 6.4.
6. Compilare la tabella di verità e stabilire le modalità per la correzione nell'addizione di due numeri in BCD aventi $S > 1001_2 = 9_{10}$.
7. Descrivere le operazioni di scalamento aritmetico a destra e a sinistra richieste per la normalizzazione dei numeri in virgola mobile nelle operazioni aritmetiche.

7. MEMORIA CENTRALE

Questo capitolo descrive gli elementi, l'organizzazione, e il funzionamento della memoria principale (*main memory*) di un microcomputer. Sebbene la memoria centrale in linea di principio possa essere costituita con qualsiasi mezzo adatto a svolgere la funzione di memorizzazione, qui la discussione è limitata alle *memorie a semiconduttore*, largamente usate nei microcomputer.

7.1. Memorie a semiconduttore

Le memorie a semiconduttore sono delle strutture cellulari di elementi di semiconduttore, o *celle*, capaci di memorizzare una informazione binaria. Un singolo chip di semiconduttore tipicamente comprende da 256 a 16384 celle identiche. Le memorie a semiconduttore possono essere classificate secondo le applicazioni, secondo la tecnologia di fabbricazione o secondo la velocità di funzionamento.

Applicazioni delle memorie

I due tipi di memorie a semiconduttore più comunemente usate sono la *RAM* (*random-access memory*: memoria ad accesso diretto, non ordinato, non sequenziale) che è del tipo a lettura e scrittura e la *ROM* (*read-only memory*: memoria a sola lettura) usata per memorizzare informazioni fisse come programmi, sottoprogrammi e tavole di riferimento (*look-up table*). Le RAM e le ROM sono strutture regolari molto simili di celle di memoria; in una ROM però l'informazione che

stabilisce il valore binario (0 o 1) di ciascun bit può essere inserita solo durante la fabbricazione.

Durante lo sviluppo di un programma, spesso è desiderabile per l'utilizzatore poter programmare in proprio (*field-programming*) una ROM elettricamente, inserendo delle informazioni senza ricorrere al costruttore. Questa esigenza ha condotto allo sviluppo di *ROM programmabili*, o *PROM* (*programmable ROM*). Esistono due tipi fondamentali di PROM. In uno di essi, ogni cella della memoria comprende un collegamento metallico che può essere fuso durante la programmazione applicando un impulso di elevata corrente per una durata specificata; un collegamento interrotto definisce il valore binario uno, mentre il collegamento integro rappresenta l'altro valore binario. Nel secondo tipo, le celle della memoria vengono programmate stabilendo in modo selettivo dei corto circuiti per mezzo della scarica a valanga.

In una ROM programmabile, la struttura di una cella risulta alterata in modo irreversibile dalla programmazione; in una *ROM riprogrammabile* invece si ha in più la possibilità di cambiare la decisione precedente. In questo tipo di ROM (*EPROM*: *erasable programmable ROM*, ROM programmabile cancellabile), l'informazione è inserita attraverso un processo a valanga, e può essere cancellata mediante l'applicazione di luce ultravioletta attraverso una finestra di quarzo, consentendo in tal modo la riprogrammazione. Questa sequenza può essere ripetuta diverse volte senza degradare le prestazioni della memoria.

Tecnologia delle memorie a semiconduttore

Le celle di memoria possono essere ottenute con transistori ad effetto di campo a metallo-ossido-silicio (MOSFET) o con transistori bipolari. In seguito al progressivo sviluppo, il numero di transistori richiesti per una cella di memoria è andato diminuendo da 8 ad 1. Qui si descrive il funzionamento della cella con 3 transistori MOS a canale *n* di fig. 7.1.

La cella è costituita dai transistori Q_1, Q_2, Q_3 e dalla capacità C_G; essa è di tipo *dinamico*, perché l'informazione viene affidata alla capacità del gate di un FET, e non invece ad uno stato stabile ottenuto mediante reazione positiva tra i componenti attivi.

Per scrivere entro una cella, si pone al livello logico 1 la linea WRITE SELECT, che è comune a tutte le celle di una riga della matrice; questa operazione mette in conduzione il transistore Q_1 e trasfe-

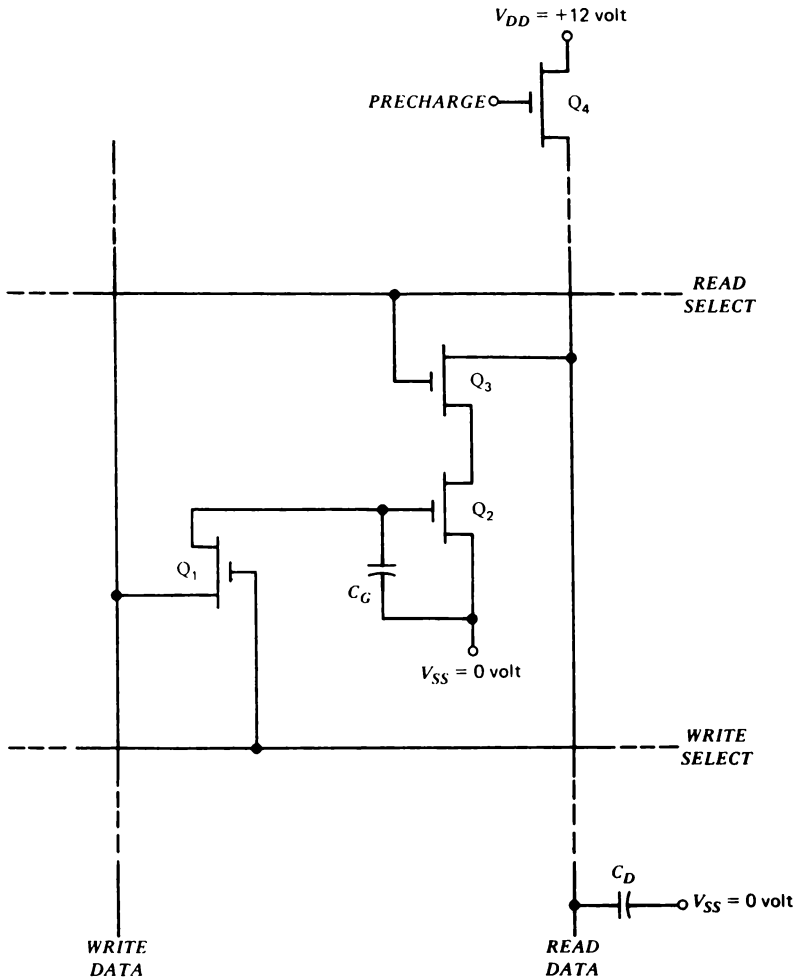



FIG. 7.1. - Cella di memoria dinamica con tre transistori MOS a canale *n*.

risce alla capacità C_G il livello logico della linea WRITE DATA, che è comune a tutte le celle di una colonna della matrice.

Per leggere il contenuto della cella di memoria, prima la capacità C_D viene caricata ad una tensione prossima a V_{DD} attraverso il transistore Q_4, comandato dalla linea *PRECHARGE*, comune a tutte le celle di una *colonna* della matrice; poi la linea *READ SELECT*, che è comune a tutte le celle di una *riga* della matrice, viene posta al livello logico 1,

ossia a $V_{DD} = + 12$ V. A questo punto, se la tensione ai capi di C_G inizialmente è sopra la tensione di soglia di Q_2, C_D si scarica attraverso Q_3 e Q_2; viceversa, se la tensione ai capi di C_G inizialmente è sotto la tensione di soglia Q_2, C_D resta carica ad una tensione prossima a V_{DD}. In questo modo, l'informazione binaria complementare a quella contenuta in C_G è trasferita alla capacità C_D lungo la linea *READ DATA* senza alterare lo stato di C_G.

Sebbene la lettura non sia distruttiva, la carica della capacità C_G tende a disperdersi, per cui l'informazione contenuta in ciascuna cella della memoria deve essere riscritta periodicamente.

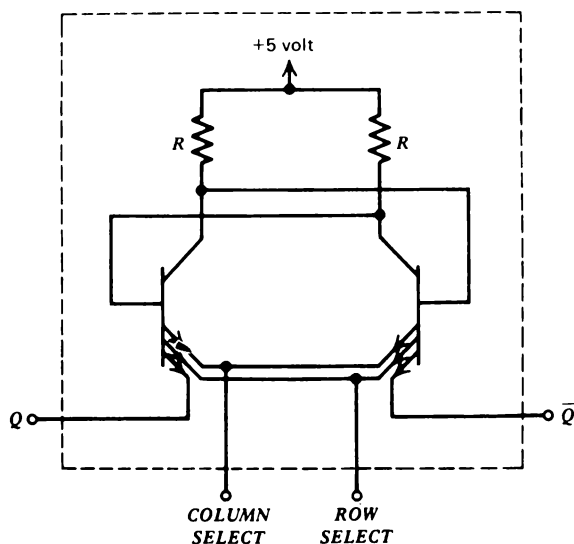



FIG. 7.2. - Cella di memoria bipolare con transistori ad emettitore triplo e indirizzamento a coincidenza.

L'operazione, che vien detta *rinfrescamento* (*refreshing*), consiste nel leggere il contenuto di una cella e nel riscriverlo immediatamente dopo nella stessa cella. I circuiti che svolgono questa attività sono discussi nel paragrafo 7.5.

Le memorie bipolari normalmente sono invece di tipo *statico*; vale a dire, ogni cella è costituita da due invertitori connessi in reazione positiva in modo da originare due stati stabili. In Fig. 7.2 è mostrata una cella di memoria bipolare che utilizza transistori con tre emettitori. Per le operazioni di scrittura o di lettura, le linee *CO-*

LUMN SELECT e *ROW SELECT* devono essere poste contemporaneamente allo stato logico alto. In scrittura, l'informazione è introdotta nella cella forzando nello stato binario basso la linea Q oppure la linea $\overline{Q}$, dipendentemente dal valore binario da memorizzare; in lettura, si « sente » (*sense*) la corrente che fluisce in queste linee (fig. 7.3).

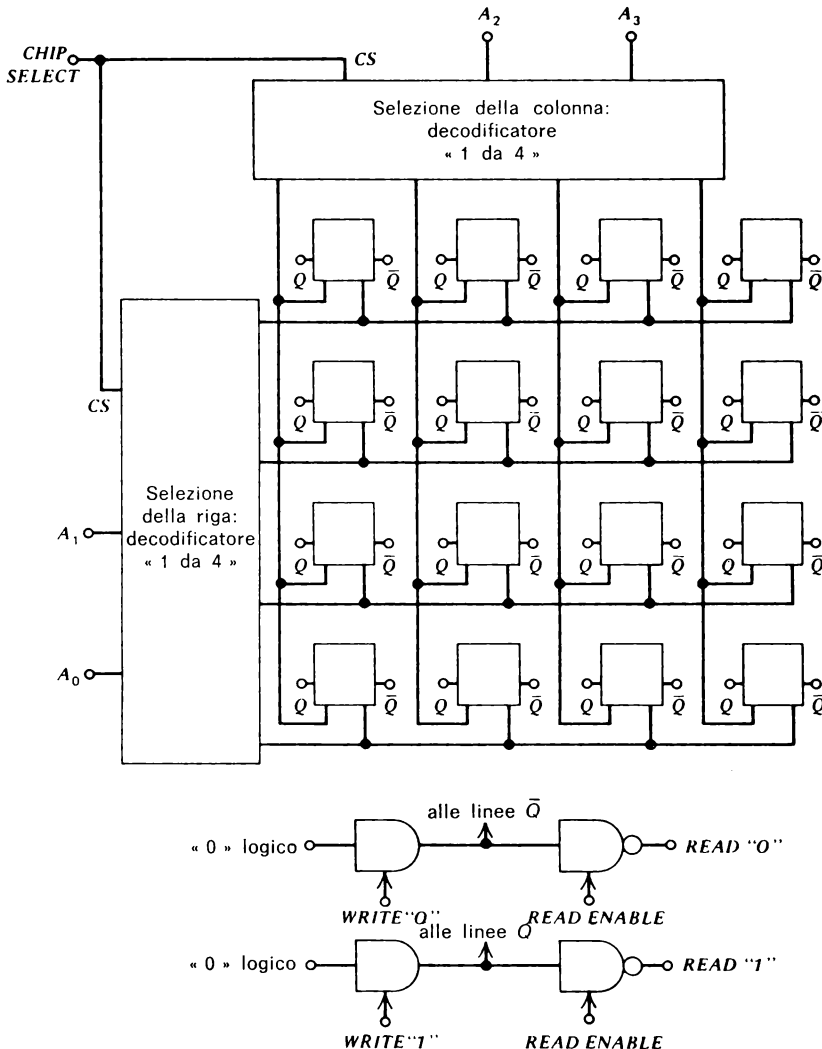



FIG. 7.3. - Schema funzionale di una matrice RAM a 16 bit con indirizzamento a coincidenza. Ogni cella è costituita come in Fig. 7.2.

L'uso delle celle di memoria di fig. 7.2 è illustrata in fig. 7.3 per il caso di una matrice per 16 bit. Essa comprende gli ingressi per l'indirizzamento, da A_0 fino a A_3, ed un ingresso *CS* (*CHIP SELECT*) che abilita la matrice e consente l'impiego di più matrici nella stessa memoria. Le uscite *READ* « *0* » e *READ* « *1* », del tipo *tristate*, sono abilitate da un normale segnale *READ ENABLE* che consente di collegarle in parallelo con le uscite analoghe di altre matrici.

Velocità di funzionamento

La velocità di funzionamento di una memoria può essere espressa mediante parametri diversi. Uno di questi è il *tempo d'accesso* (*access time*), che è il tempo che intercorre fra l'applicazione di un indirizzo e un'uscita valida dalla memoria; i valori tipici sono compresi fra alcuni nanosecondi e 2 microsecondi, e dipendono dalla tecnologia e dal numero di parole della memoria. Un altro parametro è il *tempo di ciclo* (*cycle time*), che è il tempo minimo richiesto fra l'inizio di successive operazioni di *lettura*, di *scrittura* o di *lettura-modificazione-scrittura*.

In generale le memorie MOS sono più lente di quelle bipolari; il più piccolo tempo d'accesso ottenibile con le memorie bipolari è circa 5 nanosecondi.

7.2. Organizzazione della memoria

L'organizzazione di una RAM bipolare che impiega celle di memoria a tre emettitori è stata mostrata in fig. 7.3. Quando non si richieda la maggior velocità di funzionamento propria dei circuiti bipolari, si possono usare i circuiti MOS più lenti, ma a densità più elevata. In fig. 7.4 è mostrato il diagramma a blocchi di una RAM a MOS per 4096 bit. Ciascun bit della RAM può essere indirizzato individualmente (*organizzazione a bit*: *bit organization*). I 12 bit dell'indirizzo A_{11} — A_0 sono decodificati dal *decodificatore di riga* e dal *decodificatore di colonna*, dando luogo a due gruppi di 64 linee; un bit è selezionato mediante coincidenza fra una linea di uscita del decodificatore di riga e una linea di uscita del decodificatore di colonna.

Mentre l'organizzazione a bit è prevalente nelle RAM più grandi, l'*organizzazione a parole* (*word organization*) è usata di preferenza

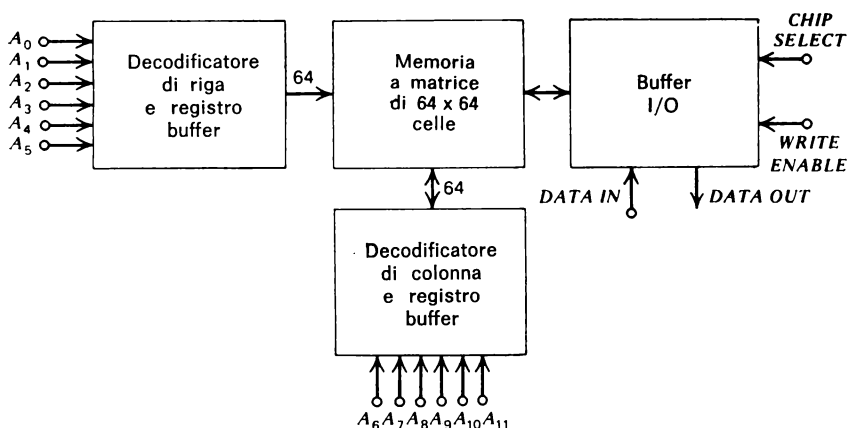



FIG. 7.4. - Schema a blocchi semplificato di una RAM a MOS di 4096 parole di 1 bit.

nelle RAM e nelle ROM di minori dimensioni. In una memoria organizzata a parole, un *decodificatore dell'indirizzo* (*address decoder*) seleziona una parola con lunghezza di diversi bit, anziché di uno solo.

7.3. Registri a scorrimento

Oltre alle memorie dei tipi discussi finora, un microcomputer può anche utilizzare *registri a scorrimento* (*shift register*). Un registro a scorrimento è costituito da una struttura regolare lineare (*linear array*) di elementi di memoria: l'informazione contenuta nel registro è fatta scorrere dall'*orologio di scorrimento* (*shift clock*), che normalmente è un orologio a due o a quattro fasi; il senso dello scorrimento è fisso oppure stabilito dal controllo di *senso di scorrimento* (*shift direction*). L'informazione può essere inserita e rimossa in corrispondenza delle estremità dell'*array* lineare; in alcuni registri a scorrimento è possibile introdurre o estrarre informazioni anche in corrispondenza di altri punti lungo il registro.

Per via della loro struttura iterativa combinata con l'elevata densità dei componenti, i registri a scorrimento forniscono un mezzo poco costoso per la memorizzazione dell'informazione.* Si possono facil-

* Una descrizione dettagliata del funzionamento dei registri a scorrimento è riportata in Bibliografia 1.

mente ottenere capacità di 2 × 1024 bit su un singolo chip di semiconduttore; in un altro tipo di registro a scorrimento, che ricorre alla tecnica dei *dispositivi ad accoppiamento di carica* (*CCD*: *charge-coupled device*), sono disponibili fino a 16384 bit su un unico chip di semiconduttore.

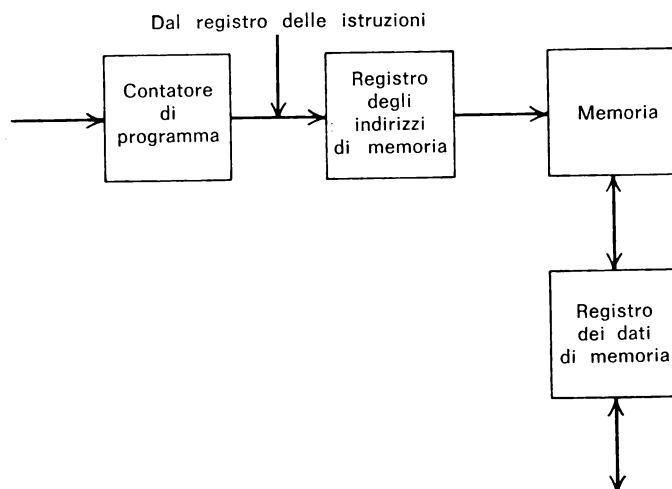



FIG. 7.5. - Schema a blocchi semplificato di un'unità di memoria con contatore di programma, registro degli indirizzi di memoria e registro dei dati di memoria.

7.4. Registri ausiliari

Un flusso ordinato di informazioni da e per la memoria centrale richiede, in aggiunta a questa, diversi registri ausiliari, ed inoltre canali (*bus*) per il transito dei dati e degli indirizzi. Un diagramma a blocchi che comprende alcuni registri ausiliari è mostrato in fig. 7.5. Il *registro dei dati della memoria* (*MDR*: *memory data register*) è uno strumento essenziale per il trasferimento dei dati da e per la memoria centrale. Il *registro degli indirizzi della memoria* (*MAR*: *memory address register*) è in grado di indirizzare individualmente ciascuna parola della memoria centrale; le sue informazioni gli derivano dal *registro delle istruzioni* (*instruction register*) o dal *contatore delle istruzioni* o *contatore di programma* (*program counter*) che contiene l'indirizzo della successiva istruzione da prelevare in memoria e che normalmente è incrementato di uno dopo che questo indirizzo è stato trasferito nel MAR.

7.5. Circuito di rinfrescamento per RAM dinamiche a MOS

Si è visto che in una cella di memoria dinamica a MOS (fig. 7.1), l'informazione è affidata ad una piccola capacità, normalmente molto minore di 1 picofarad. Tale memorizzazione volatile è soggetta a perdita di informazione, specialmente alle temperature elevate, perché la corrente di fuga della capacità raddoppia per ogni aumento di temperatura di circa 10 °C. Per questa ragione si deve ricorrere alla ricarica periodica della capacità (*refreshing*: *rinfrescamento*). Le specifiche relative al massimo intervallo di tempo ammesso fra due successive operazioni di rinfrescamento sono prudenziali e richiedono un rinfrescamento ogni pochi millisecondi.

In fig. 7.6 è illustrato un circuito di rinfrescamento per una memoria organizzata attorno a un RAM a MOS di 4096 parole di 1 bit. I 4096 bit sono disposti secondo 64 (= 2^6) righe e 64 colonne. Poiché normalmente basta applicare i segnali di rinfrescamento solo alle righe di una RAM, l'operazione è sequenzializzata da un *contatore di rinfrescamento* (*refresh counter*) di 6 bit e scandita da un *temporizzatore di*

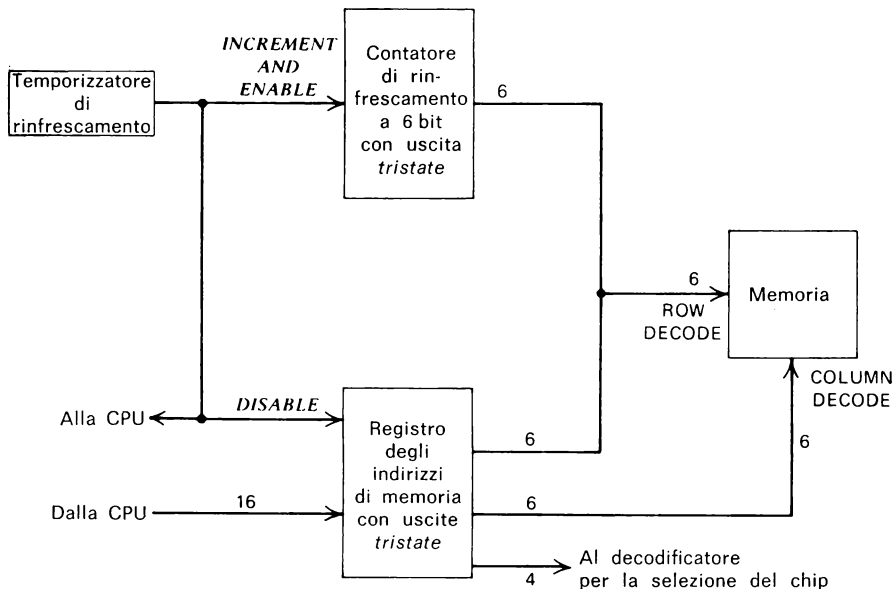



FIG. 7.6. - Circuito di rinfrescamento *cycle steal* per una RAM dinamica a MOS costituita da una matrice di 4096 parole di 1 bit. Si noti che le linee degli indirizzi e quelle dei dati della CPU devono essere disabilitate durante il rinfrescamento.

rinfrescamento (*refresh timer*), che fornisce un impulso ogni 30 microsecondi, in modo che ognuna delle 64 righe venga rinfrescata a intervalli regolari di tempo di circa 2 millisecondi. Il rinfrescamento dura un intero ciclo di macchina (*cycle steal*: *ruba un ciclo*); durante questo ciclo si sospende il funzionamento della CPU (*CPU freezing*: *congelamento della CPU*), si disabilita il registro degli indirizzi della memoria, le cui uscite passono nello stato di alta impedenza, e si abilita il contatore di rinfrescamento.

In alcuni casi è impossibile anche il rinfrescamento mediante l'applicazione periodica di un unico impulso che rinfresca l'intera memoria.

7.6. Modi di indirizzamento

In molti microcomputer la lunghezza delle istruzioni è sufficiente a contenere l'indirizzo di ogni locazione di memoria.

> **Esempio 7.1.** Un microcomputer ha parole lunghe 8 bit e $2^{16} = 65636$ locazioni di memoria. L'istruzione « load accumulator » è costituita da un operatore di 8 bit che individua l'istruzione, seguito da un operando di 16 bit che specifica l'indirizzo di memoria dal quale devono essere presi i dati da caricare nell'accumulatore.

In altri microcomputer, invece, il formato delle istruzioni che fanno riferimento alla memoria ha lunghezza insufficiente e non consente l'indirizzamento di tutta la memoria. Questa limitazione ha dato origine a vari *modi di indirizzamento* (*addressing mode*). Nella maggior parte di essi, l'*indirizzo effettivo* (*effective address*) viene calcolato combinando la parte dell'istruzione relativa all'indirizzo con un'altra parola che precedentemente è stata introdotta in un registro oppure in una locazione di memoria specificata; sono quindi necessarie due parole per indicare completamente l'indirizzo di una locazione di memoria. Con riferimento al formato d'istruzione di 16 bit mostrato in fig. 7.7, vengono ora illustrati quattro di questi modi di indirizzamento. Nel caso di figura, i 6 bit più significativi sono riservati all'operatore, mentre gli 8 bit meno significativi rappresentano la *parte indirizzo* (*address field*), o *parte scostamento* (*displacement field*), che con-

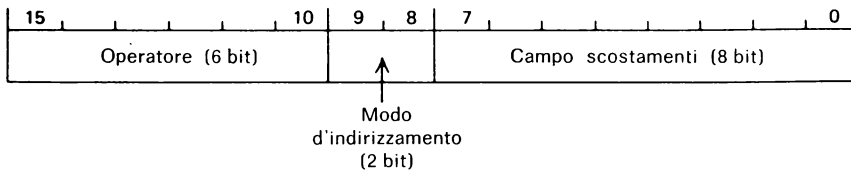



FIG. 7.7. - Parola d'istruzione a 16 bit di un microcomputer con quattro differenti modi d'indirizzamento.

sente l'indirizzamento di 256 (= 2^8) locazioni di memoria. Gli altri due bit, e precisamente i bit 8 e 9, indicati come *bit del modo di indirizzamento,* distinguono i quattro modi di indirizzamento.

Indirizzamento in pagina base

L'indirizzamento in pagina base (*base page addressing*) è il più semplice dei modi di indirizzamento; esso è noto anche come *indirizzamento in pagina zero* (*addressing on page zero*).

> **Esempio 7.2.** Si suppone che i bit 8 e 9 di fig. 7.7 siano 00, indicando l'indirizzamento in pagina base. L'intervallo di parole di memoria che può essere indirizzato con questo schema è compreso fra 0 e 255.

Indirizzamento con registro di pagina

Lo schema di indirizzamento in pagina base descritto precedentemente risulta sufficiente solo per i sistemi di memoria più piccoli. Per aumentare la capacità di indirizzamento come richiesto dalle memorie più grandi, si suddivide la memoria in *pagine* e si istituisce un *registro di pagina* (*page register*). Come la pagina base, ogni altra pagina della memoria contiene solamente 256 parole; tuttavia, il registro di pagina consente al programma di conoscere la pagina, ossia l'insieme di 256 parole, da usare in ogni dato indirizzamento alla memoria. In effetti, il registro di pagina aggiunge bit di ordine più elevato alla parte dell'indirizzo contenuta nell'istruzione, permettendo così l'indirizzamento

di altre locazioni di memoria come illustrato in fig. 7.8. Sebbene ora si richieda un'istruzione supplementare per predisporre il registro di pagina, lo schema può risultare interessante se il programmatore ha cura di effettuare la maggior parte degli indirizzamenti alla memoria all'interno di una stessa pagina.

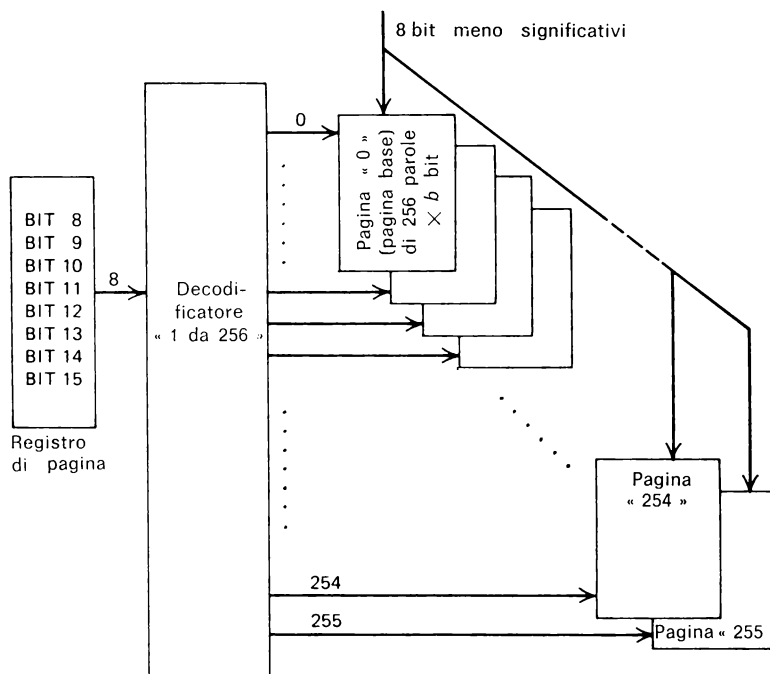



FIG. 7.8. - Indirizzamento mediante il registro di pagina. Ciascuna delle 256 pagine viene selezionata per mezzo del registro di pagina; ogni pagina contiene 256 parole.

Esempio 7.3. Supponiamo che i bit 8 e 9 di fig. 7.7 siano 01, indicando indirizzamento con registro di pagina. Il registro di pagina di fig. 7.8 è caricato con 10000001 ($= 129_{10}$), che costituiscono i bit di ordine più elevato di un indirizzo effettivo a 16 bit; di conseguenza, il decodificatore seleziona la pagina 129_{10}. Gli indirizzi ai quali si può accedere includendo gli 8 bit di ordine più basso sono quindi compresi nell'intervallo da 129×2^8 a $129 \times 2^8 + 255$, ossia compresi fra 33024_{11} e 33279_{10}.

Indirizzamento relativo al contatore di programma

In questo modo di indirizzamento, gli 8 bit meno significativi dell'istruzione sono riguardati come un numero binario con segno, nel quale il bit 7 assume la funzione di bit del segno. L'indirizzo effettivo è ottenuto aggiungendo al contenuto (*PC*) del contatore di programma (*program counter*) il numero con segno così formato. Questo consente l'indirizzamento relativo entro l'intervallo compreso fra $(PC) - 128_{10}$ a $(PC) + 127_{10}$. Se il contatore di programma è già stato incrementato al momento in cui si forma l'indirizzo e sta quindi puntando all'istruzione successiva, l'intervallo di indirizzamento è da $(PC) - 127_{10}$ a $(PC) + 128_{10}$.

L'indirizzamento in pagina base e l'indirizzamento relativo al contatore di programma sono economici perché non richiedono istruzioni supplementari per indicare la pagina. Sebbene l'intervallo di indirizzamento che essi consentono sia modesto, con una accurata programmazione si riescono ad effettuare numerosi indirizzamenti entro di esso. Inoltre, l'indirizzamento relativo al contatore di programma semplifica la *rilocazione* del programma, che serve a combinare differenti segmenti del programma e sottoprogrammi di biblioteca, residenti in memoria.

Indirizzamento relativo a un registro indice

In questo modo di indirizzamento, l'indirizzo effettivo è ottenuto come somma del contenuto di un *registro indice* (*index register*) e della parte scostamento dell'istruzione. Il vantaggio di usare registri indice sta nella loro capacità di essere modificati (cioè cancellati, caricati, incrementati o decrementati) in una frazione del ciclo di memoria, per cui il cambiamento dell'indice è una maniera molto semplice per indirizzare in successione i differenti elementi di un *array* o di altre strutture di dati.

7.6. Indirizzamento indiretto

Indirizzamento indiretto significa che la locazione all'indirizzo stabilito con uno dei modi di indirizzamento discussi finora non contiene

l'operando desiderato, ma solamente il suo indirizzo. In altri termini, l'istruzione di riferimento alla memoria indirizza a una locazione di memoria il cui contenuto serve esclusivamente come *puntatore* (*pointer*) verso l'indirizzo desiderato. Normalmente l'istruzione contiene un bit che indica se l'indirizzamento è diretto o indiretto.

Esempio 7.4. L'indirizzamento indiretto è illustrato in fig. 7.9. In questo caso, a differenza di quello di fig. 7.7, la parola di istruzione è costituita da un operatore a 7 bit, da un bit D/I che indica il modo di indirizzamento diretto o indiretto, e da un *campo scostamenti* di 8 bit. Supponiamo allora che il bit D/I del registro istruzioni indichi indirizzamento indiretto; in tal caso, quando il MAR porta l'indirizzo 1012, il contenuto della locazione 1012, ossia 10128, è interpretato come indirizzo della locazione in cui si trova l'operando.

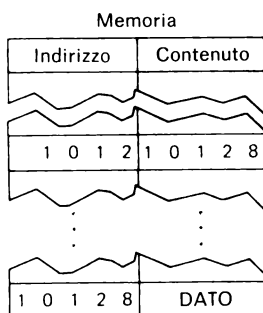
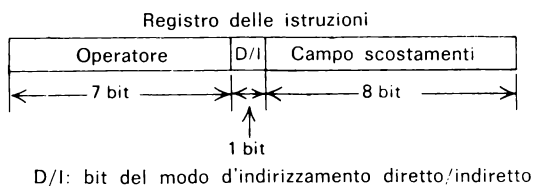



FIG. 7.9. - Illustrazione dell'indirizzamento indiretto.

Problemi

1. Quante RAM di 256 parole di 1 bit sono richieste per un sistema di memoria di 1024 parole di 8 bit? Tracciare un diagramma

a blocchi del sistema includendo decodificatori di indirizzo per la selezione del chip.

2. Dato un registro a scorrimento a ricircolazione seriale per 1024 bit, come quello discusso nel paragrafo 7.3, costituisce una memoria ricircolante per 1024 parole di 8 bit. Indicare i circuiti digitali supplementari richiesti per determinare la posizione del bit in corrispondenza del quale si può cancellare l'informazione o si possono inserire nuovi dati.
3. Discutere i vantaggi relativi delle RAM e dei registri a scorrimento e descrivere applicazioni di entrambi i tipi di memoria.
4. Discutere i meriti relativi dell'indirizzamento con registro di pagina e dell'indirizzamento relativo al contatore di programma. Mostrare un esempio di ciascuno di essi.
5. Nell'indirizzamento con registro di pagina mostrato in fig. 7.8, il registro di pagina è posto a 10100100 e i bit di ordine inferiore sono 01011011. Quale pagina verrà scelta e quale locazione di memoria verrà indirizzata?

8. UNITÀ DI CONTROLLO

L'unità di controllo è il *centro nervoso* del microcomputer; essa fornisce la sequenzializzazione e la temporizzazione delle attività, e controlla la formazione delle vie per il transito dei dati fra le varie parti del microcomputer.

8.1. Sequenzializzazione

In un microcomputer una istruzione viene eseguita in uno o più *cicli dell'elaboratore* (*processor cycle*), indicati anche come *cicli di macchina* (*machine cycle*), *microcicli* (*microcycle*), o *cicli base di istruzione* (*basic instruction cycle*). Un ciclo di macchina comprende due *fasi* principali: la *fase di prelievo* o *di indirizzamento* (*fetch* o *addressing phase*) e la *fase di esecuzione* (*execute phase*). La fase di prelievo è una fase di accesso alla memoria, durante la quale la locazione di memoria indirizzata attraverso il registro degli indirizzi di memoria (MAR: memory address register) viene messa in comunicazione con il registro dei dati di memoria (MDR: memory data register), con il quale scambia i dati. Nei casi più semplici, una istruzione è prelevata in una singola fase di prelievo ed eseguita in una singola fase di esecuzione.

> **Esempio 8.1.** Nell'Esempio 2.3, l'addizione è eseguita sommando il contenuto del registro B al contenuto dell'accumulatore e ponendo il risultato nell'accumulatore. L'istruzione per questa addizione occupa una sola locazione di memoria, per cui può essere prelevata in una sola fase di prelievo; inoltre, siccome essa coinvolge solamente la CPU e non richiede

altri accessi alla memoria, può essere eseguita in una sola fase d'esecuzione. Di conseguenza, questa istruzione è completata in un'unica fase di prelievo e in un'unica fase d'esecuzione, quindi in un solo ciclo di macchina.

Le istruzioni, specialmente nei microcomputer che hanno parole di lunghezza minore di 16 bit, possono occupare però più di una locazione di memoria.

Esempio 8.2. Un microcomputer ha parole di 8 bit e ha la memoria centrale costituita da $2^{16} = 65536$ parole. La memorizzazione di un'istruzione di salto (*jump*) richiede tre locazioni di memoria: la prima locazione contiene il codice dell'operazione di salto in linguaggio di macchina; la seconda e la terza locazione contengono l'indirizzo a 16 bit che specifica la locazione di memoria alla quale deve saltare il programma.

Un'istruzione che occupa più locazioni di memoria, per essere prelevata integralmente richiede altrettante fasi d'accesso, e quindi altrettanti cicli di macchina. Siccome però normalmente l'esecuzione di un'istruzione non può cominciare finché l'intera istruzione non è stata trasferita nel *registro delle istruzioni* (*instruction register*), le fasi di esecuzione sono inutilizzate o sono omesse dai cicli di macchina finché non sono terminate tutte le fasi di prelievo richieste.

Quando, come nell'Esempio 8.1, l'istruzione prelevata non richiede altri accessi alla memoria, essa viene eseguita durante la fase d'esecuzione immediatamente successiva alla fase di prelievo che completa il trasferimento dell'istruzione nel registro delle istruzioni; quando invece l'esecuzione dell'istruzione prelevata richiede altri accessi alla memoria, si deve ricorrere ad altrettanti cicli di macchina supplementari.

Esempio 8.3. Questo esempio illustra la successione delle fasi di prelievo e di esecuzione e la loro funzione nel caso di una istruzione d'addizione. Il microcomputer di questo esempio, simile a quello di fig. 8.4, ha parole di 8 bit e memoria centrale con capacità di $2^{16} = 65536$ parole.

La Tab. 8.1 mostra il contenuto delle locazioni della memoria centrale interessate dalle operazioni. Nelle locazioni di

TABELLA 8.1

Contenuto delle locazioni significative della memoria centrale durante l'elaborazione dell'istruzione d'addizione dell'esempio 8.3

Indirizzo		Contenuto	
Decimale	Binario	Binario	Esadecimale
0	0000000000000000	????????	??
1	0000000000000001	????????	??
.	.	.	.
.	.	.	.
.	.	.	.
101	0000000001100101	11000001	C1
102	0000000001100110	00000100	04
103	0000000001100111	10110101	B5
.	.	.	.
.	.	.	.
.	.	.	.
1205	0000010010110101	00001111	0F
.	.	.	.
.	.	.	.
.	.	.	.
65534	1111111111111110	????????	??
65535	1111111111111111	????????	??

memoria 101, 102 e 103 è immagazzinata l'istruzione « Somma il contenuto della locazione di memoria 1205 al contenuto dell'accumulatore e poni il risultato nell'accumulatore ». La locazione 101 contiene il codice di operazione per l'addizione, che è 11000001 ($= C1_{16}$) nel linguaggio di macchina del microcomputer. La locazione 102 contiene gli 8 bit più significativi dell'equivalente binario 1205, la locazione 103 gli 8 bit meno significativi. La locazione 1205 contiene l'addendo 00001111 = $= 15_{10} = F_{16}$.

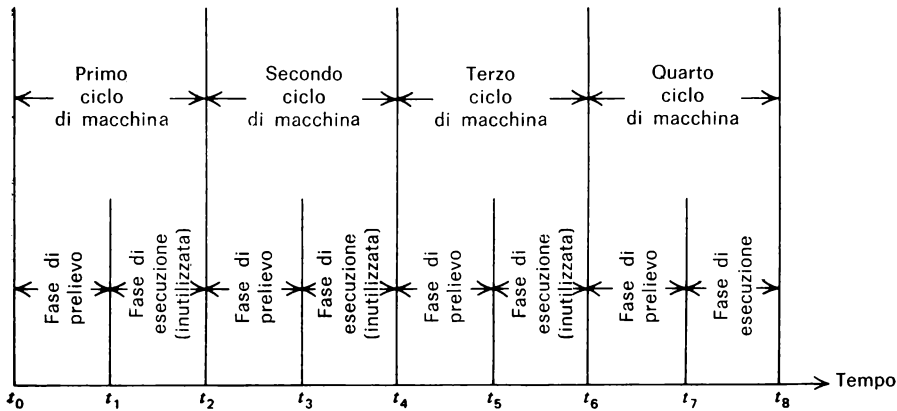



FIG. 8.1. - Elaborazione dell'istruzione di somma dell'esempio 8.3.; sequenza dei cicli di macchina e delle fasi di accesso e di esecuzione.

La successione dei cicli di macchina e delle fasi di prelievo e di esecuzione è mostrata nella fig. 8.1. Nella Tab. 8.2 è riportato il contenuto dell'accumulatore, del registro degli indirizzi di memoria, del registro dei dati di memoria e del registro delle istruzioni nel corso dell'esecuzione di questa istruzione.

Durante la fase di prelievo del primo ciclo di macchina, cioè da t_0 a t_1, attraverso il MAR a 16 bit viene indirizzata la locazione di memoria 101 e attraverso MDR il codice di operazione $1100000 = C1_{16}$ viene ricopiato da questa locazione nei primi 8 bit del registro delle istruzioni. L'istruzione non è completa, e di conseguenza la fase d'esecuzione nel primo ciclo di macchina (da t_1 a t_2) è inutilizzata.

Durante la fase di prelievo del secondo ciclo di macchina (da t_2 a t_3), gli 8 bit più significativi dell'indirizzo 1205 sono ricopiati dalla locazione di memoria 102 nei secondi 8 bit del registro delle istruzioni. L'istruzione non è ancora completa, per cui anche la fase d'esecuzione del secondo ciclo di macchina (da t_3 a t_4) è inutilizzata.

Durante la fase di prelievo del terzo ciclo di macchina (da t_4 a t_5), gli 8 bit meno significativi dell'indirizzo 1205 sono prelevati dalla locazione di memoria 103 e poi trascritti nei terzi 8 bit del registro delle istruzioni. Ora l'istruzione contenuta

TABELLA 8.2

Contenuto dell'accumulatore, del MAR, del MDR e del registro delle istruzioni fra gli istanti t_0 e t_8 durante l'elaborazione dell'istruzione d'addizione dell'esempio 8.3

Ciclo di macchina	Fase	Istante	Accumulatore		MAR		MDR		Registro delle istruzioni	
			Binario	Decimale	Binario	Decimale	Binario	Esadecimale	Binario	Esadecimale
1	Accesso	t_0	00011000	24	0000000001100101	101	????????	??	????????????????????????	??????
	Esecuzione	t_1	00011000	24	0000000001100101	101	11000001	C1	11000001????????????????	C1????
2	Accesso	t_2	00011000	24	0000000001100110	102	11000001	C1	11000001????????????????	C1????
	Esecuzione	t_3	00011000	24	0000000001100110	102	00000100	04	1100000100000100????????	C104??
3	Accesso	t_4	00011000	24	0000000001100111	103	00000100	04	1100000100000100????????	C104??
	Esecuzione	t_5	00011000	24	0000000001100111	103	10110101	B5	110000010000010010110101	C104B5
4	Accesso	t_6	00011000	34	0000010010110101	1205	10110101	B5	110000010000010010110101	C104B5
	Esecuzione	t_7	00011000	24	0000010010110101	1205	00001111	0F	110000010000010010110101	C104B5
		t_8	00100111	39	????????????????	????	00001111	0F	110000010000010010110101	C104B5

nel registro delle istruzioni è completa e indica che si deve prelevare il contenuto della locazione di memoria 1205. La fase d'esecuzione del terzo ciclo di macchina (da t_5 a t_6) è ancora inutilizzata.

Durante la fase di prelievo del quarto ciclo di macchina, cioè da t_6 a t_7, viene indirizzata la locazione di memoria 1205, e il suo contenuto è ricopiato nel MDR. L'addizione è eseguita durante la fase d'esecuzione (da t_7 a t_8) di questo ciclo, sommando il contenuto del MDR, che è 15, al contenuto dell'accumulatore, che è 24, e ponendo nell'accumulatore il risultato, che è 39.

L'accesso alla memoria può corrispondere al trasferimento di una istruzione o di parte di una istruzione, oppure al trasferimento di dati dalla memoria principale al MDR, come nel caso dell'Esempio 8.3; inversamente, l'accesso alla memoria può anche corrispondere al trasferimento di dati dal MDR alla memoria centrale.

Esempio 8.4. Un microcomputer ha parole di 8 bit e ha memoria centrale costituita da $2^{16} = 65536$ parole. Nelle locazioni di memoria 201, 202 e 203 è memorizzata l'istruzione « Memorizza il contenuto dell'accumulatore nella locazione di memoria 1305 ». La locazione 201 contiene il codice di operazione per la memorizzazione, che è 00001101 nel linguaggio di macchina del microcomputer; la locazione 202 contiene gli 8 bit più significativi dell'equivalente binario di 1305, la locazione 203 gli 8 bit meno significativi. Il contenuto dell'accumulatore è $11000011 = 195_{10}$.

L'istruzione di memorizzazione trasferisce il numero 195_{10} dall'accumulatore alla locazione di memoria 1305 attraverso il MDR, distruggendo in questo processo il precedente contenuto della locazione 1305. Il contenuto dell'accumulatore resta invariato, vale a dire resta 195_{10}.

Il trasferimento di dati dal MDR alla memoria principale ha luogo durante la fase di accesso alla memoria, anche se, come nell'Esempio 8.4, esso costituisce in effetti l'esecuzione dell'istruzione.

Esempio 8.5. La sequenzializzazione dell'istruzione di memorizzazione dell'Esempio 8.4 è illustrata in Figura 8.2. L'istru-

zione è prelevata durante le prime tre fasi di accesso alla memoria; l'esecuzione dell'istruzione, cioè la memorizzazione del numero 195_{10} nella locazione di memoria 1305, è eseguita durante la quarta fase di accesso.

La fase di accesso frequentemente viene prolungata aggiungendo una *fase di attesa* (*wait phase*) ausiliaria, nota anche come *stato d'attesa* (*wait state*), quando il *tempo d'accesso* (*access time*) alla locazione di memoria indirizzata supera la durata della fase di accesso. La

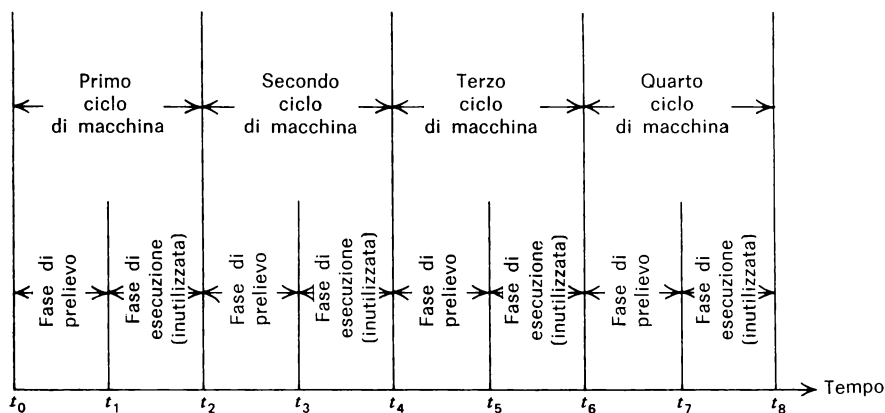



FIG. 8.2. - Sequenza dei cicli di macchina e delle fasi di accesso e di esecuzione relativi all'esempio 8.5.

durata della fase d'attesa viene scelta abbastanza lunga d'assicurare il completamento del trasferimento dei dati fra il registro MDR e la locazione di memoria indirizzata.

Un'altra fase ausiliaria è la *fase d'arresto* (*halt phase*), nota anche come *stato d'arresto* (*halt state*), durante la quale il funzionamento della CPU è sospeso; le memorie dinamiche continuano però ad essere rinfrescate, se è necessario. La fase d'arresto, che viene iniziata da una istruzione d'arresto oppure da un segnale su una speciale linea d'arresto, viene terminata o mediante un *interrupt* o mediante un intervento manuale attraverso i controlli del pannello frontale.

8.2. Temporizzazione

Un ciclo di macchina comprende un numero intero di *stati* (*time state*) di durata uguale, normalmente compresa fra 0,1 e 3 microsecondi. Questa durata è stabilita dall'*orologio del sistema* (*system clock*), ed è quindi costante in un dato microcomputer. Il numero di stati che compongono una fase d'esecuzione, una fase d'attesa o una fase d'arresto può variare, mentre è normalmente costante il numero degli stati che compongono una fase di accesso.

Esempio 8.6. Un microcomputer con parole di 8 bit ha memoria centrale costituita da $2^{16} = 65536$ parole. L'indirizzamento delle locazioni di memoria impiega 16 bit che sono inviati dalla CPU alla memoria principale mediante un bus di dati da 8 bit durante due stati temporali consecutivi. In questo modo la fase di accesso risulta costituita da due stati temporali; essa può essere seguita da una fase d'attesa costituita da un numero qualsiasi di stati temporali.

8.3. Vie dei dati e struttura a bus

Dati e istruzioni vengono scambiati fra le varie parti di un microcomputer attraverso le *vie dei dati* (*data path*). I collegamenti si sviluppano lungo i *bus*, che sono fasci di linee, una per ogni bit trasmesso in parallelo, ai quali possono accedere i sottosistemi del microcomputer. Le vie dei dati vengono costituite facendo accedere ad un bus o fra di loro i sottosistemi che devono comunicare. Se il flusso delle informazioni è unidirezionale, come fra MAR e memoria centrale, l'accesso è controllato da una porta AND per ciascuna linea del bus; se invece il flusso è bidirezionale, allora l'accesso è controllato da un circuito simile a quello di fig. 4.1.

Il funzionamento di un bus è illustrato nell'esempio che segue.

Esempio 8.7. La fig. 8.3 illustra la trasmissione dei dati fra tre registri per parole di 4 bit. Le uscite dei flip-flop *D-E* sono collegate al bus dei dati attraverso circuiti logici *tristate*, che vengono attivati dai segnali *SEND*. Gli ingressi *D* dei flip-flop

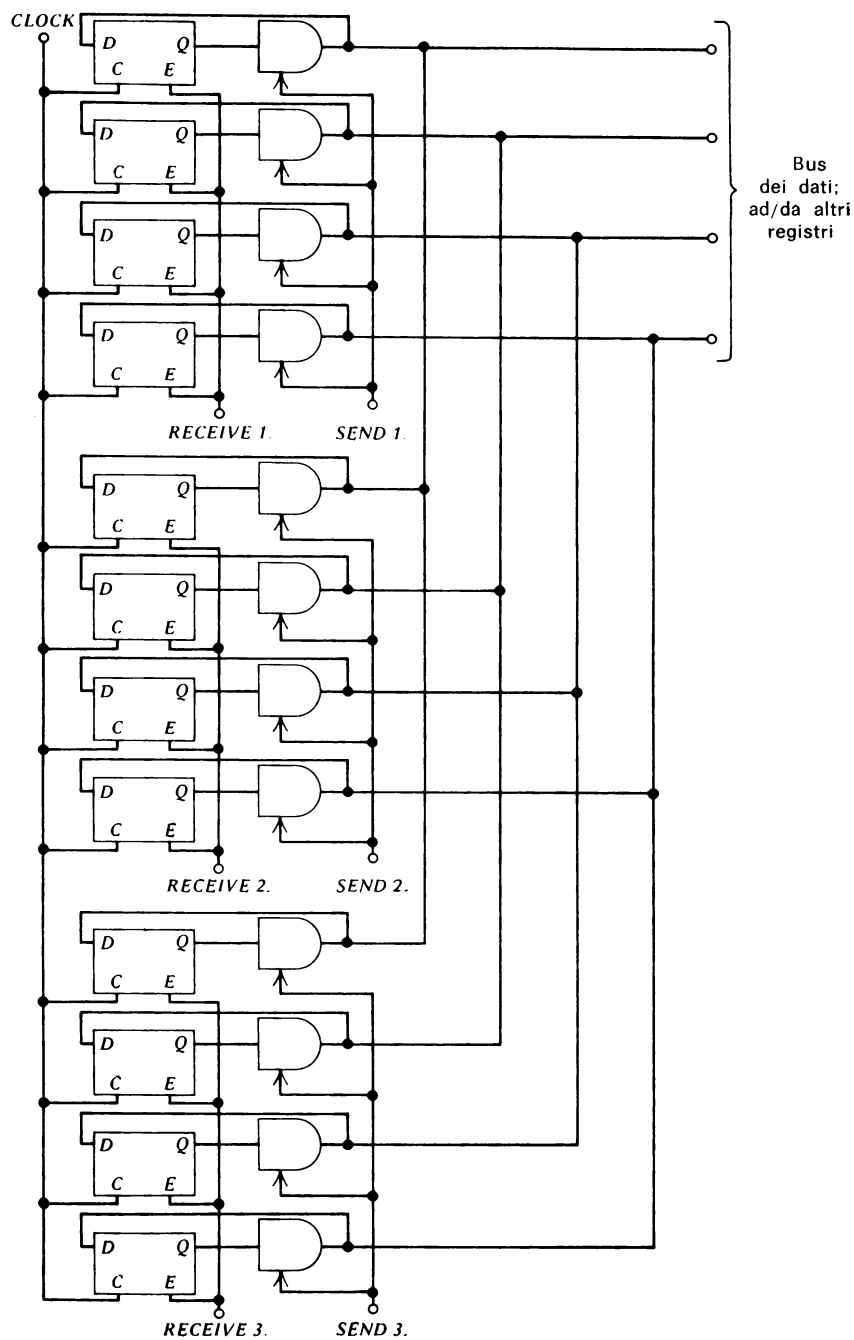



FIG. 8.3. - Struttura a bus per la formazione delle vie dei dati fra tre registri da 4 bit.

D-E sono anch'essi collegati al bus dei dati e sono attivati dai segnali *RECEIVE*. In ogni istante, possono essere attivi un solo segnale *SEND* e un numero qualsiasi di segnali *RECEIVE*. La fig. 8.3 illustra il funzionamento del bus dei dati con tre registri; è però possibile collegare al bus anche altri registri.

Durante l'elaborazione di un'istruzione, l'unità di controllo costituisce le vie dei dati richieste per prelevare l'istruzione dalla memoria centrale e ricopiarla nel registro delle istruzioni. L'istruzione viene poi decodificata dal *decodificatore delle istruzioni* (*instruction decoder*) che segnala all'unità di controllo le successive vie dei dati da costituire.

Esempio 8.8. Nell'Esempio 8.3, nell'istante t_0 (vedi fig. 8.1 e Tav. 8.2) vengono costituite le vie dei dati richieste per prelevare i primi 8 bit dell'istruzione. Questi 8 bit sono poi decodificati dal decodificatore delle istruzioni nell'istante t_1; essi indicano che l'attuazione della istruzione richiederà tre ulteriori fasi di prelievo ed una fase di esecuzione. Le vie dei dati per queste fasi saranno costituite negli istanti t_2, t_4, t_6, e t_7, rispettivamente.

8.4. Microprogrammazione

Le vie dei dati e la successione della loro formazione sono stabilite dal *corredo di istruzioni* del microcomputer (*instruction set*). In questo modo, una volta scelto il set delle istruzioni e implementati nella circuiteria (*hardware*) del microcomputer il decodificatore delle istruzioni e i circuiti che controllano la formazione delle vie dei dati, non è più possibile modificare o espandere il set delle istruzioni. Una soluzione più flessibile si ottiene realizzando la decodificazione delle istruzioni e il controllo per le vie dei dati mediante *firmware*, ossia ricorrenlo ad una ROM; il risultato è un microcomputer *microprogrammato*. È necessario considerare che il termine microprogrammazione (*microprogramming*) non è un'abbreviazione di programmazione di un microcomputer, ma descrive un metodo per la formazione delle vie dei dati.

Nella microprogrammazione, le vie dei dati relative ad un'istruzione in linguaggio di macchina sono attivate da un *microprogramma* (*microprogram*), quelle relative ad un ciclo di macchina (*microcycle*) sono attivate da una *microistruzione* (*microinstruction*), e quelle relative ad uno stato temporale sono attivate da una *microoperazione* (*micro-operation*). In questo modo un microprogramma risulta costituito da una o più microistruzioni, e una microistruzione da una o più microoperazioni.

In linea di principio, ogni istruzione di un microcomputer microprogrammato potrebbe essere rappresentata da un microprogramma memorizzato in una ROM; in pratica però questo approccio richiederebbe ROM di una grandezza impossibile. La situazione può essere superata sostituendo la ROM con una *PLA* (*programmable logic array*: *array logico programmabile*), che è una forma di matrice più elaborata. Secondo un altro approccio, le sequenze di microoperazioni che intervengono in numerose microistruzioni sono memorizzate una sola volta, sotto forma di un sottoprogramma microprogrammato, o *microroutine*.

> **Esempio 8.9.** Tutti i cicli di macchina che richiedono il trasferimento di una parola dalla memoria centrale al MDR nella fase di prelievo, utilizzano le stesse vie dei dati. Le informazioni per costituire in sequenza le vie dei dati richieste durante questa fase sono memorizzate in una *microroutine*.

La circuiteria necessaria per la microprogrammazione comprende una ROM che viene programmata (*masked*) per un particolare insieme di istruzioni. La ROM fornisce il *controllo delle vie dei dati* che attiva le vie dei dati di volta in volta richieste, e il *controllo dell'indirizzo successivo*, che determina la locazione della microoperazione successiva. I circuiti logici esterni alla ROM comprendono un *controllo dell'indirizzo della ROM* e un *registro indirizzi della ROM*.

8.5. Schema a blocchi di un microcomputer

In fig. 8.4 è riportato lo schema a blocchi di un possibile microcomputer; è semplificato, ma mostra ugualmente alcuni dettagli della CPU. Si tenga presente che lo schema considerato presenta una certa

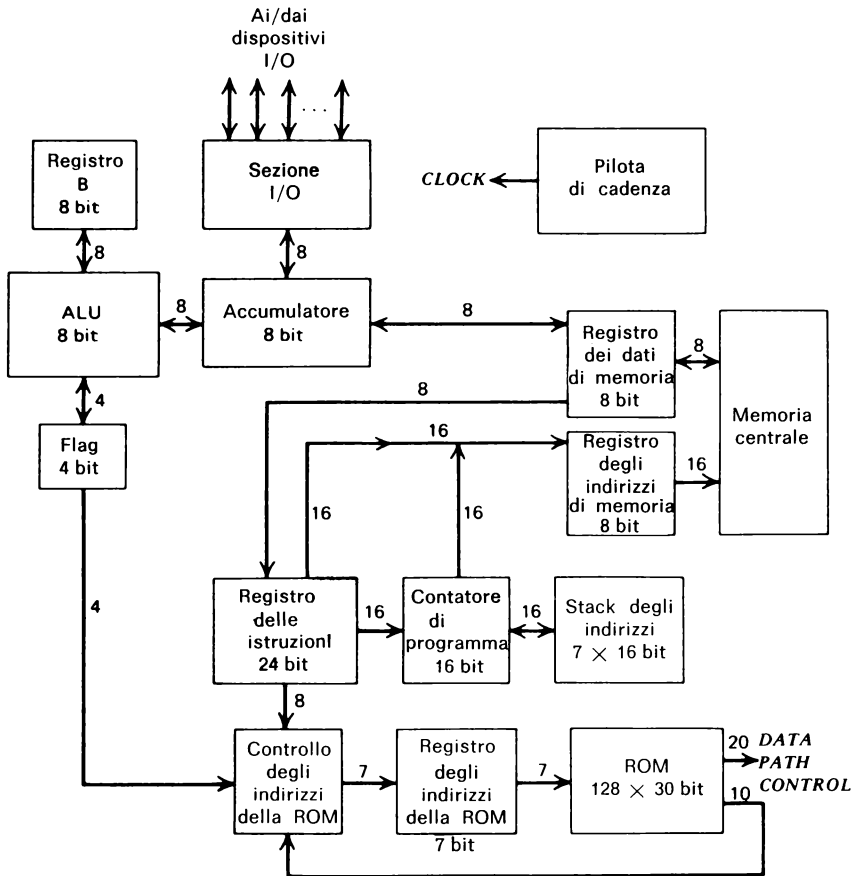



FIG. 8.4. - Schema a blocchi semplificato di un microcomputer.

arbitrarietà, perché l'architettura di un microcomputer può variare molto.

La parola di memoria è lunga 8 bit, per cui la ALU, l'accumulatore, il registro B e il registro MDR hanno tutti capacità di 8 bit. Il MAR invece è a 16 bit, per cui può indirizzare $2^{16} = 65536$ locazioni di memoria; il contatore di programma e ciascuno dei 7 livelli dello *stack* (cfr. Cap. 9) hanno quindi anch'essi capacità di 16 bit. Infine, il registro delle istruzioni ha capacità di 24 bit: 8 bit per l'operatore e 16 bit per l'operando. Le istruzioni vengono trasferite a questo registro dalla memoria centrale attraverso il registro MDR in tre gruppi di 8 bit (*byte*).

La memoria centrale, descritta nel Cap. 7, comunica con la CPU attraverso il registro MDR bidirezionale, ed è indirizzata dal MAR, che può essere caricato sia dal contatore di programma sia dalla zona del registro delle istruzioni riservata all'operando.

La CPU a sua volta comunica con l'esterno attraverso la sezione I/O, descritta nel Cap. 4. Le istruzioni aritmetiche e logiche sono attuate dalla ALU, dall'accumulatore, dal registro B e da 4 flag.

L'esecuzione delle istruzioni è ordinata da una ROM microprogrammata con la capacità di $2^7 = 128$ parole di 30 bit, che predispone a questo scopo le vie dei dati e l'assetto della ALU e della sezione I/O, mediante le uscite *DATA PATH CONTROL*.

La ROM è governata da una propria unità di controllo (*ROM address control*), che combina le informazioni fornite dai flag, dalla zona del registro delle istruzioni riservate all'operatore e dall'indicazione di indirizzo successivo fornita dalla ROM stessa. La circuiteria ausiliaria della ROM comprende anche il registro degli indirizzi, a 7 bit.

Problemi

1. Quale delle istruzioni del paragrafo 3.1 può essere eseguita in un solo ciclo di macchina?
2. Sostituire i punti interrogativi di corrispondenza di t_0 e t_8 nella Tav. 8.2 con appropriati valori 1 o 0 nel caso in cui l'istruzione è preceduta o seguita da un'istruzione identica.
3. Come deve essere cambiata la Tab. 8.2 se la locazione di memoria 1205 contiene 21 anziché 15?
4. Preparare una tabella che mostri il contenuto delle locazioni significative della memoria centrale prima e dopo l'attuazione dell'istruzione descrittta negli Esempi 8.4 e 8.5.
5. Compilare una tabella che mostri il contenuto dell'accumulatore, del MAR, del MDR e del registro delle istruzioni dall'istante t_0 fino all'istante t_8 nel caso dell'istruzione descritta negli Esempi 8.4 e 8.5.
6. Nel microcomputer dell'Esempio 8.6 è necessaria una fase d'attesa se il tempo d'accesso alla memoria è pari a (*a*) 0.5 volte lo stato temporale, (*b*) 2.5 volte lo stato temporale?

7. Trovare il tempo richiesto per attuare l'istruzione descritta nell'Esempio 8.3 se la durata di una fase di prelievo è quella di una fase di esecuzione 1 1 microsecondo, e se il tempo d'accesso alla memoria è minore di 1 microsecondo. Ripetere nel caso in cui siano omesse le fasi d'esecuzione non necessarie.
8. Stabilire il valore binario dei segnali *RECEIVE* e *SEND* richiesti in fig. 8.3 per caricare il contenuto del registro superiore negli altri due registri. Supporre che per l'attivazione sia richiesto un valore binario 1.
9. Descrivere tutte le vie dei dati che vengono attivate durante l'attuazione dell'istruzione d'addizione dell'Esempio 8.3.
10. Nelle istruzioni del paragrafo 3.1 elencare quelle fasi di prelievo e di esecuzione che in un microcomputer microprogrammato sarebbero realizzate da una microroutine.

9. COMPLEMENTI DI PROGRAMMAZIONE

Questo capitolo presenta alcuni aspetti del *software* più avanzati di quelli discussi nel Cap. 3. Vengono qui descritti i programmi assemblatori e di caricamento, le strutture di dati, i collegamenti di sottoprogramma, i simulatori, e il sistema operativo.

9.1. Assemblatori

Un programma *assemblatore* (*assembler*), o *assemblatore simbolico* (*symbolic assembler*) traduce un programma da un linguaggio assemblativo in un linguaggio di macchina. Esso inoltre assegna il posto in memoria alle istruzioni, alle variabili e alle costanti. Ciascuna istruzione in linguaggio assemblativo è tradotta in una istruzione in linguaggio di macchina, ad eccezione delle *macroistruzioni* e delle *pseudo istruzioni.* Le pseudo istruzioni comprendono l'assegnazione di valori specifici (per esempio, EQUATE $W = 1.5$), la definizione di macroistruzioni (per esempio, DEFINE MACRO MAC1), la specificazione di porte I/O, varie istruzioni di partenza (*start*) e di fine (*end*), specificazioni per la memorizzazione di blocchi di dati, e specificazioni di formato.

L'azione dell'assemblatore è illustrata con un programma relativo alla frase dell'Esempio 3.7: « IF $V > W$ THEN $X \leftarrow Y - 1$ ELSE $X \leftarrow Y + 1$ ». In fig. 9.1 è mostrato il *flow chart* del programma e nella Tab. 9.1 un programma in linguaggio assemblativo. Fatta eccezione per le pseudo istruzioni EQUATE e END, ogni istruzione è numerata

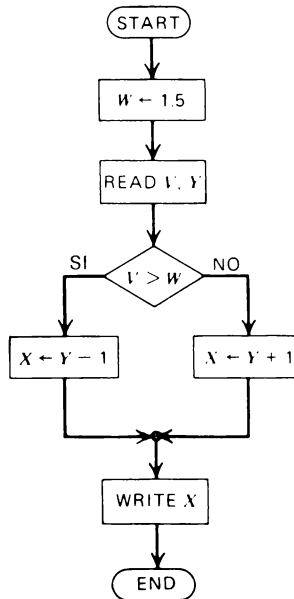



FIG. 9.1. - *Flow chart* di un programma relativo alla frase « IF $V > W$ THEN $X \leftarrow Y - 1$ ELSE $X \leftarrow Y + 1$ ».

mediante un *numero di riga* (*line number*); questi numeri di riga non sono utilizzati dall'assemblatore ma vengono riportati per facilitare il compito del programmatore. Ogni linea numerata rappresenta una istruzione singola; di conseguenza nel caso di una macroistruzione il numero delle righe corrisponde al numero delle istruzioni in linguaggio assemblativo comprese nella macroistruzione.

Esempio 9.1. Nell'esempio 3.5 la macroistruzione MAC1 è definita come segue:

DEFINE MACRO MAC1
. . .
. . . (lista di 20 istruzioni che definiscono MAC1)
. . .
END

A questa macroistruzione sono assegnati 20 numeri di riga ogni volta che è inclusa in un programma in linguaggio assemblativo.

Nella Tab. 9.1, le *label* (*etichette*) identificano gli indirizzi di destinazione per le istruzioni JUMP e GO TO, e sono tradotte in locazioni di memoria dall'assemblatore. Gli *operatori* (*operator*) sono *codici di operazione mnemonici* noti all'assemblatore, gli *operandi* (*operand*) sono variabili e costanti specificate dal programmatore e i *commenti* (*comment*), normalmente in qualche linguaggio naturale, servono ad informazione del programmatore e sono ignorati dall'assemblatore.

Gli assemblatori più comuni sono quello *ad una passata* (*one-pass*) e quello *a due passate* (*two-pass*). L'assemblatore ad una passata traduce un programma in linguaggio assemblativo in un programma in linguaggio di macchina scandendo il *programma sorgente* una volta soltanto; proprio per questo motivo, un tale assemblatore è costretto ad introdurre passi di programma supplementari quando il programma in linguaggio assemblativo fa riferimento ad una label successiva.

> **Esempio 9.2.** Il programma in linguaggio assemblativo mostrato nella Tav. 9.1 è tradotto in un programma in linguaggio di macchina da un assemblatore ad una passata. Nella riga 5 si incontra un riferimento alla label successiva L1 (numero di riga 10), che rappresenta una locazione di memoria ancora sconosciuta. Di conseguenza l'assemblatore deve cercare e trovare L1 oppure deve ritornare più tardi alla riga 5 per inserire la locazione di memoria rappresentata dalla label L1.

La manipolazione delle label è più semplice in un assemblatore a due passate. Un tale assemblatore percorre due volte il programma in linguaggio assemblativo. Durante la prima passata riconosce le label e genera una *tavola dei simboli* (*symbol table*), costituita dalle label e dalle loro locazioni di memoria. Le istruzioni in linguaggio assemblativo sono tradotte in istruzioni in linguaggio di macchina durante la seconda passata. Le informazioni generate nella prima passata spesso sono trasmesse ad una memoria esterna come un nastro di carta o un disco magnetico per l'impiego nella seconda passata.

9.2. Loader

Un programma in linguaggio di macchina viene trasferito nella memoria centrale del microcomputer da un programma *di carica-*

TABELLA 9.1

Programma in linguaggio assemblativo relativo alla fase « IF V > W THEN X ← Y − 1 ELSE X ← Y + 1 »

Numero di riga	Label	Istruzione		Commento
		Operatore	Operando	
		EQUATE	*W*, 1.5	Il valore 1.5 è assegnato a *W*
1		READ	*V*	Da un dispositivo d'ingresso viene letto il valore di *V*
2		READ	*Y*	Da un dispositivo d'ingresso viene letto in valore di *Y*
3		LOAD	*W*	Il valore di *W* è caricato nel-l'accumulatore
4		SUBTRACT	*V*	Il valore di *V* è sottratto al contenuto dell'accumulatore; il segno del risultato è indicato dal flag del segno
5		JUMP ON FLAG < 0	L1	Il programma salta alla label L1 se il flag del segno indica che il contenuto dell'accumulatore è negativo
6		LOAD	*Y*	Il valore di *Y* è caricato nel-l'accumulatore
7		ADD IMMEDIATE	1	Il numero 1 è sommato al contenuto dell' accumulatore
8		STORE	*X*	Il contenuto dell'accumulatore è memorizzato come *X*
9		GO TO	FINISH	Il programma salta alla label FINISH
10	L1	LOAD	*Y*	Il valore di *Y* è caricato nel-l'accumulatore
11		SUBTRACT IMMEDIATE	1	Il numero 1 è sottratto al contenuto dell' accumulatore
12		STORE	*X*	Il contenuto dell'accumulatore è memorizzato come *X*
13	FINISH	WRITE	*X*	Il valore di *X* viene scritto in un dispositivo d'uscita
		END		

mento (*loader*: *caricatore*). Il loader legge le istruzioni in sequenza e cerca i sottoprogrammi richiamati dal programma e già presenti, o *residenti*, in memoria (*sottoprogramma di biblioteca*: *library routine*). Alcuni loader sono in grado di *rilocare* il programma: questa proprietà è particolarmente utile quando nella memoria centrale vengono caricati due o più programmi.

Quando si fa partire il microcomputer, un piccolo loader, spesso indicato come *bootstrap loader* (*loader autoavviante*), è usato per preparare il microcomputer al caricamento nella memoria centrale del primo programma (di solito un loader più grande). Il loader *bootstrap* può essere memorizzato in una memoria permanente di vario tipo: logica cablata, ROM, nastro di carta o disco magnetico; in modo alternativo, può essere introdotto manualmente attraverso i controlli del pannello frontale.

9.3. Strutture di dati

In un microcomputer, la memorizzazione strutturata o *struttura di dati* (*data structure*) consente la conservazione e il recupero di dati *correlati* (*related*). Ogni metodo che consenta il recupero di tali dati può essere considerata una struttura di dati; nel seguito tuttavia ci si limita a considerare le *pile* (*stack*) e le *code* (*queue*), che sono entrambe memorizzazioni in locazioni di memoria consecutive. Un'altra importante struttura di dati, la matrice (*array*), è discussa brevemente nei Problemi 3 e 4 alla fine di questo capitolo.

Stack

Una delle strutture di dati più semplici, usata comunemente per i collegamenti con i sottoprogrammi, è lo *stack* (*pila*), noto anche come stack *push-down* (*spingi giù*) o come stack *LIFO* (*last-in-first-out*: *ultimo a entrare primo ad uscire*). In uno stack tutti gli accessi sono fatti in corrispondenza ad una estremità di un gruppo di locazioni di memoria consecutive. In fig. 9.2 sono mostrate tre rappresentazioni di uno stack per 7 parole. In tutti e tre i casi, la parte superiore della figura mostra la pila occupata da 3 parole, mentre la parte inferiore mostra la stessa pila accresciuta della parola 1111, aggiunta

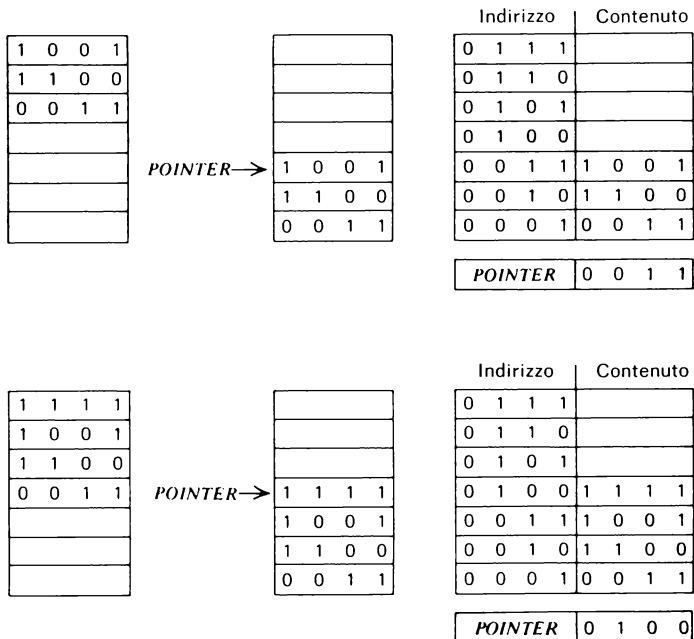



FIG. 9.2. - Tre forme di *stack* per 7 parole di 4 bit. Nella parte superiore lo *stack* è occupato da 3 parole, nella parte inferiore da 4 parole.

in sommità. Se ora si rimuove una parola, esce quella introdotta per ultima, ossia la parola 1111, e lo stack ritorna nello stato precedente, mostrato nella parte superiore di fig. 9.2. La colonna all'estrema sinistra rappresenta lo stack nella forma *push-down*. Essa ricorda la pila di piatti sostenuta da una molla che si incontra nelle *cafeterias*, nella quale il piatto superiore resta sempre al medesimo livello anche quando si aggiungono o si tolgono piatti alla pila. Questa forma di stack può essere realizzata con un registro a scorrimento bidirezionale.

Nella seconda colonna di fig. 9.2 è mostrata un'altra forma di stack. Questa ricorda una serie di libri impilati verticalmente, quando si intenda che ogni aggiunta e ogni rimozione venga fatta in corrispondenza della sommità della pila dei libri. L'altezza dello stack è indicata dal *puntatore* (*pointer*). Questa forma di stack può essere realizzata nella memoria centrale oppure in una memoria *scratch-pad* nel modo indicato nella colonna più a destra di fig. 9.2; si noti che al puntatore deve essere riservata una propria locazione di memoria.

In fig. 9.3 sono mostrati i *flow chart* relativi alle operazioni con lo stack della colonna più a destra in fig. 9.2. In fig. 9.3 *a* è mostrato il sottoprogramma per l'accrescimento dello stack (*push stack*), mentre in fig. 9.3 *b* è mostrato il sottoprogramma per il decrescimento (*pop stack*). Si noti che questi sottoprogrammi non provvedono a rivelare l'*overflow dello stack* che si verifica quando lo stack è pieno e si inizia l'introduzione di un'altra parola, ne l'*underflow dello stack* (*stack underflow*), che si verifica quando lo stack è vuoto e si inizia la rimozione di una parola. I provvedimenti per rivelare queste situazioni, che nei sistemi maggiori normalmente sono già inclusi, nei sistemi minori toccano al programmatore.

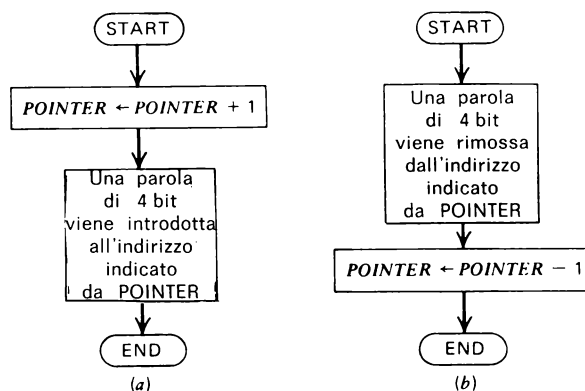



FIG. 9.3. - Programmi relativi alla gestione di uno *stack*: (*a*) accrescimento; (*b*) decrescimento.

Code

Un'altra forma di struttura di dati, usata comunemente nell'attesa di accesso ad un dispositivo di ingresso o d'uscita, è la *coda* (*queue*) d'attesa. L'accrescimento di una coda avviene in corrispondenza del suo termine (*tail*), mentre il suo decrescimento avviene in corrispondenza del suo principio (*head*).

In fig. 9.4 sono mostrate quattro forme di coda. La forma della colonna più a sinistra può essere realizzata mediante registri a scorrimento con ingresso in serie e uscita in parallelo. La forma della seconda colonna è usata raramente ed è stata inclusa solo per ragioni di completezza.

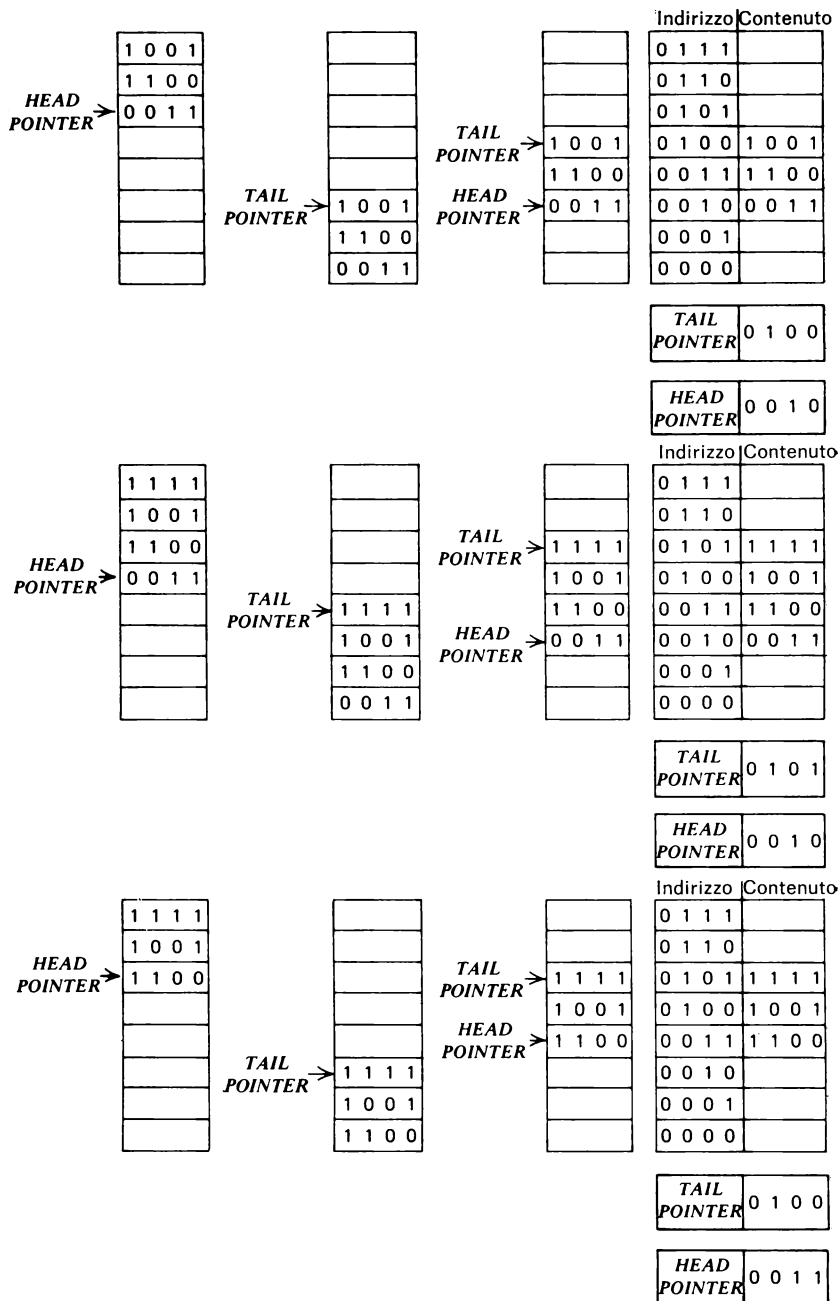



FIG. 9.4. - Quattro forme di coda per 8 parole di 4 bit. Parte superiore: la coda è occupata da 3 parole; parte centrale: la coda della parte superiore dopo l'aggiunta della parola 1111; parte inferiore: la coda della parte centrale dopo la rimozione della parola 0011.

Nella forma della terza colonna di fig. 9.4, il contenuto non è fatto scorrere. Questa forma può essere realizzata nella memoria centrale oppure in una memoria *scratch-pad* nel modo mostrato nella colonna più a destra. Si noti che l'aggiunta e la rimozione di una parola provocano il sollevamento della coda per una locazione. La coda potrebbe quindi fuoriuscire dallo spazio di memoria ad essa riservato anche quando è riempita solo parzialmente. L'inconveniente può essere evitato ripiegando la coda su se stessa, ossia dichiarando adiacenti l'indirizzo 0111 = 7_{10} e l'indirizzo 0, come è illustrato nei sottoprogrammi di fig. 9.5. Questi sottoprogrammi gestiscono la coda in modo corretto finché non si produce l'*overflow* o l'*underflow* della coda. Nei piccoli sistemi, i provvedimenti per rivelare questi incidenti restano ancora affidati alla responsabilità del programmatore, e risultano alquanto più complicati di quelli richiesti per il funzionamento di uno stack.

9.4. Collegamenti di sottoprogramma

L'accesso ad un sottoprogramma e il rientro da esso richiedono diverse operazioni indicate con il termine *collegamenti di sottoprogramma* (*subroutine linkage*). Una di queste operazioni è la memorizzazione in uno stack del contenuto del contatore di programma, da servire come *indirizzo per il ritorno* (*return address*). Il medesimo stack può anche essere utilizzato per immagazzinare il contenuto di registri e di flag, al fine di rendere questi elementi disponibili al sottoprogramma. I collegamenti di sottoprogramma provvedono anche al trasferimento di *parametri* fra programma principale e sottoprogramma.

Esempio 9.3. Questo esempio illustra l'uso dei collegamenti di sottoprogramma e il trasferimento di parametri in un programma che legge il valore di x da un nastro di carta, richiama un sottoprogramma che calcola $x + \sqrt{x}$, che richiama a sua volta un sottoprogramma che calcola $\sqrt{x}$, e stampa il risultato. Si noti che lo scopo di questo programma è di illustrare l'uso dei collegamenti di sottoprogramma, per cui il programma non è necessariamente ottimo (si veda anche il Problema 5 alla fine del capitolo). Si noti anche che

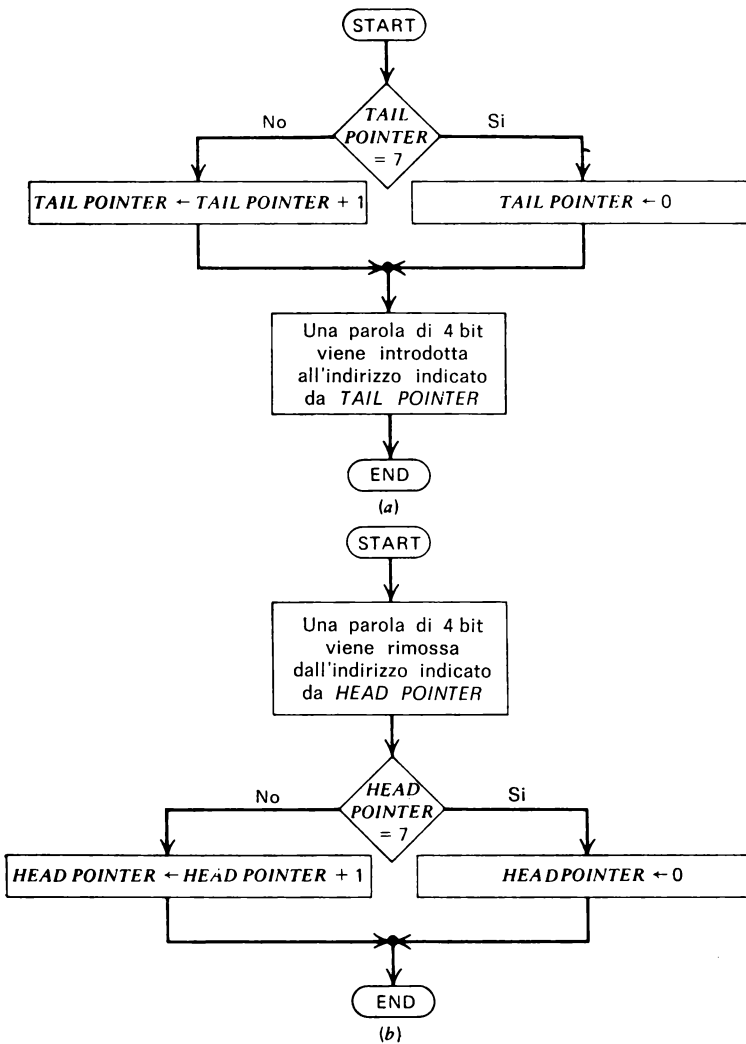



FIG. 9.5. - Programma relativo alla gestione di una coda: (*a*) accrescimento; (*b*) decrescimento.

alcune delle istruzioni in effetti in un microcomputer risultano delle macroistruzioni.

Il programma, mostrato nella Tav. 9.2, è diviso in tre parti che sono localizzate in tre aree della memoria centrale:

TABELLA 9.2.
Contenuto della memoria durante l'esecuzione del programma per il calcolo di $x + \sqrt{x}$.

LOCAZIONE DI MEMORIA	CONTENUTO
Area di memoria occupata dal programma principale	
0	Start
1	Leggi il nastro di carta e memorizza il numero nella locazione di memoria 16
2	Vai alla locazione di memoria 32 dove incomincia il sottoprogramma che calcola $x + \sqrt{x}$
3	Stampa il contenuto della locazione di memoria 17
4	Stop
Area di memoria occupata dai parametri	
16	Introduci x nel sottoprogramma che calcola $x + \sqrt{x}$
17	Esci dal sottoprogramma che calcola $x + \sqrt{x}$
18	Entra nel sottoprogramma *squareroot*
19	Esci dal sottoprogramma *squareroot*
Area di memoria occupata dai sottoprogrammi	
32	Inizio del sottoprogramma $x + \sqrt{x}$
33	Memorizza il contenuto della locazione di memoria 16 nella locazione di memoria 35
34	Vai alla locazione di memoria 36
35	Memorizza l'ingresso al sottoprogramma che calcola $x + \sqrt{x}$
36	Memorizza il contenuto della locazione di memoria 35 nella locazione 18
37	Vai alla locazione di memoria 48 dove inizia il sotto-programma *squareroot*
38	Addiziona il contenuto della locazione di memoria 19 al contenuto della locazione di memoria 35 e memorizza il risultato nella locazione di memoria 40
39	Va alla locazione di memoria 41
40	Memorizza $x + \sqrt{x}$

(*continuazione tab. 9.2*)

LOCAZIONE DI MEMORIA	CONTENUTO
41	Memorizza il contenuto della locazione di memoria 40 nella locazione di memoria 17
42	Rientra dal sottoprogramma che calcola $x + \sqrt{x}$
48	Inizio del sottoprogramma *squareroot*
49	Memorizza il contenuto della locazione di memoria 18 nella locazione di memoria 51
50	Vai alla locazione di memoria 52
51	Memorizza l'ingresso al sottoprogramma *squareroot*
52 53 54	Questi tre passi del programma calcolano la radice quadrata (*square root*) del numero memorizzato nella locazione di memoria 51. Il risultato è memorizzato nella locazione di memoria 56.
55	Vai alla locazione di memoria 57
56	Memorizza l'uscita del sottoprogramma *squareroot*
57	Memorizza il contenuto della locazione di memoria 56 nella locazione di memoria 19
58	Rientra dal sottoprogramma *squareroot*

l'*area del programma principale* (*main program area*), l'*area dei parametri* (*parameter area*) e l'*area dei sottoprogrammi* (*subroutine area*). L'area dei parametri (spesso localizzata nella pagina base) deve essere indirizzabile sia dall'area del programma principale sia dall'area dei sottoprogrammi; non è invece richiesto l'indirizzamento fra l'area del programma principale e l'area dei sottoprogrammi. La fig. 9.6 presenta l'attività del contatore di programma e dello stack; quest'ultimo è costituito da uno stack *push-down* di 2 parole del tipo mostrato nella colonna più a sinistra di fig. 9.2.

Il programma parte in corrispondenza della locazione di memoria 0 con lo stack completamente vuoto; passa alla locazione 1, esegue l'istruzione di lettura e di memorizzazione,

passa alla locazione 2 e richiama il sottoprogramma che inizia alla locazione di memoria 32. Di conseguenza, il contatore di programma assume il valore della locazione iniziale del sottoprogramma, ossia 32, mentre l'indirizzo di ritorno, ossia il valore 3, è introdotto nello stack (si vedano le prime quattro colonne da sinistra del gruppo superiore di fig. 9.6).

Ora incomincia l'esecuzione del sottoprogramma. Il parametro x che era stato memorizzato dal programma principale nella locazione 16, viene memorizzato nella locazione 35 per effetto dell'istruzione contenuta nella locazione 33. In preparazione del richiamo del sottoprogramma *squa-*

FIG. 9.6. - Contenuto del contatore di programma e dello *stack* durante l'esecuzione del calcolo $x + \sqrt{x}$.

reroot, l'istruzione della locazione 36 memorizza x nella locazione 18 dell'area dei parametri. L'istruzione della locazione 37 richiama il sottoprogramma *squareroot* che inizia alla locazione 48. Di conseguenza, il contenuto del contatore di programma è cambiato in 48, mentre l'indirizzo di ritorno 38 è introdotto nello stack (vedere le caselle più a destra nel gruppo superiore di fig. 9.6).

Si passa ora ad eseguire il sottoprogramma *squareroot*; l'esecuzione è completata con l'istruzione di ritorno della locazione 58, che introduce nel contatore del programma l'indi-

rizzo di ritorno 38 (vedi le caselle più a destra del gruppo centrale di fig. 9.6). Riprende ora l'esecuzione del primo sottoprogramma e continua fino all'istruzione di ritorno della locazione 42, che introduce nel contatore di programma l'indirizzo di ritorno 3. Viene ora eseguita l'istruzione di stampa posta alla locazione 3 del programma principale, dopo di che il programma termina con l'istruzione *Stop* della locazione di memoria 4.

9.5. Simulazione

Il comportamento di un microcomputer dovrebbe essere valutato nelle reali condizioni di funzionamento; in molti casi però può risultare utile effettuare un test simulato che non faccia ricorso ad unità I/O esterne.

Esempio 9.4. Il controller del traffico con microcomputer dell'Esempio 2.2 viene provato entro un sistema strumentale controllato da un computer più grande, senza veicoli né sensori di veicoli. I risultati ottenuti rivelano il comportamento del controller nelle condizioni di traffico pari a quelle simulate.

La simulazione può essere utilizzata anche nella fase di progetto di un sistema con microcomputer. Per esempio, si può valutare un programma simulando il microcomputer con un programma simulatore (*simulator*), che viene fatto girare in un altro computer insieme al programma da valutare.

Esempio 9.5. Si vuole valutare un sistema di controllo con microcomputer. Si scrive un programma di prova che rispetti i requisiti del sistema di controllo e le regole di programmazione del microcomputer. Questo programma e un programma simulatore vengono introdotti in un computer più grande che predice il comportamento del programma di prova e del sistema di controllo nelle condizioni di funzionamento reali.

9.6. Condivisione dell'hardware

Fin qui si è tacitamente supposto che il microcomputer e le sue unità periferiche siano dedicati ad un solo compito ed eseguano un solo programma. In realtà, non è necessario che sia sempre così; per esempio, un'unità periferica, utilizzata ad intermittenza, può essere condivisa da diversi microcomputer.

> **Esempio 9.6.** In un impianto chimico, tre processi sono controllati da tre microcomputer indipendenti ma vicini. I programmi, sviluppati altrove, sono disponibili su nastri di carta perforati. Ogni programma viene introdotto nel corrispondente microcomputer attraverso un unico lettore di nastro perforato, al quale può accedere ciascuno dei tre microcomputer.

In altre forme di condivisione dell'*hardware*, un'unica unità di memoria può essere condivisa da diversi programmi (*multiprogramming*: *multiprogrammazione*) o da diversi microprocessori (*multiple processing*: *multielaborazione*).

9.7. Funzionamento del sistema

Oltre alle unità I/O, alla CPU, alla memoria e al programma, il funzionamento di un microcomputer richiede anche l'*hardware operativo* (*operating hardware*) e il *software operativo* (*operating software*). L'hardware operativo comprende i controlli del pannello frontale e gli altri dispositivi periferici che servono come interfaccia fra il microcomputer e l'operatore umano. Il software operativo, spesso indicato come *sistema operativo* (*operating system*) o *programma esecutivo* (*executive program*), è un gruppo di programmi che agiscono in unione all'hardware operativo. Esso può comprendere un programma *editor* o *text editor*, che facilita le modifiche nel programma prima che venga assemblato, e un programma di *debugging*, che interviene durante l'esecuzione del programma per aiutare a localizzarne gli errori. Il sistema operativo provvede anche alla sequenzializzazione

o seriazione delle operazioni dell'assembler e del loader, ed inoltre governa la condivisione dell'hardware e l'esecuzione del programma.

Esempio 9.7. Si stila un programma in un linguaggio assemblativo per un microcomputer. Il risultato, indicato come *codice sorgente* (*source code, source file, source module*), può essere costituito da un *pacco di schede sorgente* (*source deck*) o da un *nastro sorgente* (*source tape*). Il codice sorgente viene introdotto nel microcomputer dove è tradotto in linguaggio di macchina dall'assemblatore. Il risultato costituisce il *codice oggetto* (*object code, object file, object module*), che può essere costituito da un *pacco di schede oggetto* (*object deck*) oppure da un *nastro oggetto* (*object tape*). Il codice oggetto viene introdotto nel microcomputer dove il programma *linkage editor* lo collega con i sottoprogrammi di biblioteca richiamati. Il risultato è un *load module* che viene caricato nella memoria centrale dal programma di caricamento (loader). Queste operazioni sono dirette dal sistema operativo e dallintervento manuale dell'operatore.

Problemi

1. Estendere il flow chart di fig. 9.3 *a* in modo da comprendere un'indicazione di overflow dello stack quando questo è pieno e si inizia l'introduzione di un'altra parola.
2. Estendere il flow chart di fig. 9.5 *a* e 9.5 *b* in modo da comprendere una indicazione di overflow della coda quando la coda è piena e si inizia l'addizione di una parola. Supporre che inizialmente la coda sia vuota ed entrambi i puntatori (pointer) puntino alla locazione 0.
3. Una forma di struttura di dati largamente usata che impiega locazioni di memoria consecutive è l'*array*. L'array più semplice è l'array ad una dimensione che si riduce ad una lista ordinata. Un esempio di array del genere è una lista contenente l'ora del tramonto per i 31 giorni di ogni mese. Preparare un flow chart che consenta di memorizzare e di leggere ciascun *elemento* di questo array. Includere un controllo che

segnala l'errore quando si inizia l'accesso ad una locazione di memoria esterna ai limiti dell'array (per esempio accesso al giorno 32).

4. Estendere il flow chart del Problema 3 precedente ad un array bidimensionale, contenente l'ora dell'alba e del tramonto per i 31 giorni di un mese.
5. Nel programma della Tab. 9.2, considerare l'area dei parametri e fornire la spiegazione della complessità delle operazioni nelle locazioni di memoria da 33 a 36 e da 49 a 51.

APPENDICE A
TAVOLE ARITMETICHE IN BASE 8

TAVOLA A.1 **Addizione**

+	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

TAVOLA A.3 **Multipli di 10_{10}**

×	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

TAVOLA A.3 **Multipli di 10_{10}**

BASE 10	BASE 8
10	12
20	24
30	36
40	50
50	62
60	74
70	106
80	120
90	132

APPENDICE B
TAVOLE ARITMETICHE IN BASE 16

TAVOLA B.1 **Addizione**

×	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	IC	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

TAVOLA B.2 **Moltiplicazione**

×	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

APPENDICE C
TAVOLA DELLE POTENZE DI 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1024	10	0.00097 65625
2048	11	0.00048 82812 5
4096	12	0.00024 41406 25
8192	13	0.00012 20703 125
16384	14	0.00006 10351 5625
32768	15	0.00003 05175 78125
65536	16	0.00001 52587 89062 5
1 31072	17	0.00000 76293 94531 25
2 62144	18	0.00000 38146 97265 625
5 24288	19	0.00000 19073 48632 8125
10 48576	20	0.00000 09536 74316 40625
20 97152	21	0.00000 04768 37158 20312 5
41 94304	22	0.00000 02384 18579 10156 25
83 88608	23	0.00000 01192 09289 55078 125
167 77216	24	0.00000 00596 04644 77539 0625
335 54432	25	0.00000 00298 02322 38769 53125
671 08864	26	0.00000 00149 01161 19384 76562 5
1342 17728	27	0.00000 00074 50580 59692 38281 25
2684 35456	28	0.00000 00037 25290 29846 19140 625
5368 70912	29	0.00000 00018 62645 14923 09570 3125
10737 41824	30	0.00000 00009 31322 57461 54785 15625
21474 83648	31	0.00000 00004 65661 28730 77392 57812 5
42949 67296	32	0.00000 00002 32830 64365 38696 28906 25

BIBLIOGRAFIA

1. BARNA, A. and PORAT, D. I., *Integrated Circuits in Digital Electronics*, Wiley-Interscience, New York, 1973.
2. FLORES, I., *Peripheral Devices*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1973.
3. HUSSON, S. S., *Microprogramming, Principles and Applications*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1970.
4. KNUTH, D. E., *Fundamental Algorithms, The Art of Computer Programming*, Addison-Wesley Publishing Company, Reading, Mass., 1968.
5. KORN, G. A., *Minicomputers for Engineers and Scientists*, McGraw-Hill Book Company, New York, 1973.
6. STONE, H. S., *Introduction to Computer Organization and Data Structures*, Mc Graw-Hill Book Company, New York, 1972.

SOLUZIONE DI ALCUNI PROBLEMI

Capitolo 2

1: 30 più il ritorno di massa

Capitolo 3

1: 00000101, 00001011, 11111111. 7: 51.

Capitolo 4

1: Da destra a sinistra.

Capitolo 5

1: 45, 0.71875, 6.625. 2: 1111011, 0.01001, 11100000.011.
7: 1 1001 0011.0010; 1 0001 0111.0010 0101; 1.0011 0110 1001.
9: $E = 0\ 1000$, $M = 0\ 111101$; $E = 0\ 0101$, $M = 1\ 01000101$; $E = 0\ 0100$, $M = 0\ 01000001$; $E = 1\ 1011$, $M\ 1\ 01011$.

Capitolo 7

1: 32. 5: $41\,984_{10}$, $42\,075_{10}$.

Capitolo 8

2: OF_{16}, $C104B5_{16}$ (5 volte), 101_{10}. 6: No, si. 7: 8 microsecondi, 5 microsecondi. 8: *RECEIVE 1* = 0, *SEND 1* = 1, *RECEIVE 2* = 1, *SEND 2* = 0, *RECEIVE 3* = 1, *SEND 3* = 0.

Capitolo 9

5: La minimizzazione dell'uso dell'indirizzamento indiretto.

INDICE ANALITICO

GLOSSARIO

Access time, 10, 68, 84
Tempo d'accesso. È l'intervallo di tempo che separa il momento in cui l'unità di controllo impartisce l'ordine di accedere ad una locazione di memoria e il momento in cui inizia il trasferimento dell'informazione a/o da quella locazione.

Accumulator, 6, 10
Accumulatore,. Registro interno all'unità centrale, nel quale viene posto uno degli operandi al momento dell'operazione e nel quale si forma il risultato.

Adder, binary, 52
Addizionatore binario
full a., 53; a. completo,
ripple a., 53; a. ad onda o a propagazione del riporto,
parallel a., 53; a. in parallelo,
look-ahead carry a., simultaneous carry a., 54; a. a riporto simultaneo.

Address, 10
Indirizzo. Elemento che individua la posizione, la destinazione o l'origine di un'informazione. Il linguaggio di macchina è costituito da un numero binario (*absolute a., machine a.;* indirizzo assoluto o indirizzo di macchina) che il progettista del computer fa corrispondere in maniera permanente e definitiva ad ogni posizione fisica nell'ambito della macchina. In un linguaggio diverso, è un gruppo di caratteri alfanumerici (*symbolic a.*; indirizzo simbolico) scelto dal programmatore. È compito del programma traduttore assegnare ad ogni indirizzo simbolico un corrispondente indirizzo assoluto.

Addressing
Indirizzamentto o identificazione dell'indirizzo. Può avvenire con differenti modalità (*a. mode*, 72); innanzi tutto si distingue fra *direct a.* (i. diretto), quando il campo operandi dell'istruzione contiene l'indirizzo, e *indirect a.* (i. indiretto), quando tale campo contiene l'indirizzo dell'indirizzo, ossia contiene l'indirizzo di una locazione di memoria che a sua volta contiene l'indirizzo. Nell'ambito di questa distinzione primaria, si distingue ulteriormente fra *relative a.* (i. relativo) oppure no. Nel caso dell'indirizzamento relativo, il campo operandi è suddiviso in due sottocampi, la base (*base*) e lo scostamento (*displacement*); l'indirizzo (diretto o indiretto) è ottenuto sommando lo spostamento al contenuto della locazione o del registro individuato dalla base, che viene quindi a costituire l'indirizzo dell'*indirizzo di riferimento* (*base address;* indirizzo di base).

Arithmetic-logic unit (ALU), 60
Unità logico-aritmetica; cfr. computer.

Array, 88, 96
Strutura ordinata. Può presentare un ordinamento semplice (*a. unidimensionale*), e allora è una lista, un ordinamento duplice (*a. bidimensionale* o *matrice*) o un ordinamento ancora superiore.

Assembler, program, 15
Assemblatore. Programma che sostituisce ogni istruzione in linguaggio assemblativo con l'istruzione o il gruppo di istruzioni corrispondente in linguaggio di macchina, e ogni indirizzo simbolico con un corrispondente indirizzo assoluto. Può risiedere in permanenza entro lo stesso computer (*resident assembler*), oppure può essere situato entro un altro computer, normalmente di maggiori dimensioni (cross-assembler).

Bit, 6
Contrazione di *binary digit,* cifra binaria. Indica una veriabile a due valori o binaria.

check bit, 46; bit di controllo, nei codici a rivelazione e correzione d'errore,

sign bit, 38; bit del segno, nella rappresentazione dei numeri negativi.

Bit slice, 62
Letteralmente, *fetta di bit.* Si riferisce al caso di un'unità aritmetica ottenuta affiancando diversi moduli integrati simili fino ad ottenere la capacità richiesta dalla lunghezza della parola da elaborare. In pratica, è il caso dei microprocessori non monolitici realizzati con la tecnologia bipolare.

Bootstrap program, 96
Programma di lancio iniziale, autoavviante. È un programma caricatore in grado di caricare anche se stesso (*bootstrap loader*).

Branching, 1
Diramazione. È la scelta fra diversi segmenti del programma, ossia fra gruppi di operazioni alternativi; è una possibilità caratteristica del computer. Comporta l'abbandono dell'esecuzione sequenziale delle istruzioni e il passaggio all'esecuzione di un'istruzione non immediatamente successiva in memoria all'istruzione che specifica la diramazione. Per questa ragione viene anche indicata come *salto* (*jump*). Può essere condizionata (*conditional b.*) al prodursi di un avvenimento determinato, oppure non condiziontta (*unconditional b.*). Fa parte delle istruzioni di tipo logico.

Buffer, 5, 24
Memoria tampone, o di transito, o intermedia. È un registro in cui vengono memorizzate le informazioni per il tempo che intercorre fra il momento in cui si rendono disponibili e il momento in cui vengono utilizzate. Svolge un ruolo essenziale nello scambio di dati fra due sistemi non sincronizzati, consentendo a ciascuno di essi di svolgere la propria attività indipendentemente dall'altro.

Bus structure, 85
Struttura a bus. Struttura a moduli connessi ad un gruppo di linee che costituiscono un canale generale di collegamento, lungo il quale si sviluppano i percorsi per il transito delle informazioni fra i vari moduli (*data path*).

Central process unit (CPU), 6
Unità centrale; comprende l'unità logico-aritmetica e l'unità di controllo; cfr. computer.

Clock, 25
Oscillatore elettronico che fornisce impulsi a cadenza fissa per sincronizzare le attività del computer.

Code
Codice.

Compiler, program, 18
Compilatore. Programma che a partire da un programma sorgente espresso in un linguaggio compilativo produce un programma oggetto contenente le operazioni e gli indirizzi assoluti in codice di macchina.

Computer
Elaboratore. Macchina elettronica automatica, sincronizzata da un oscillatore interno, a programma memorizzato, in

grado di scegliere fra differenti segmenti del programma (*branching*). Sottosistemi funzionali essenziali sono: la memoria centrale (*main memory*), dove viene conservato il programma; l'unità logico-aritmetica, che effettua le operazioni indicate dal programma; l'unità di controllo, che preleva le istruzioni in memoria, le decodifica e dispone i circuiti della ALU per l'esecuzione delle operazioni; l'unità di ingresso/uscita, per la comunicazione con l'esterno. A quest'ultima fanno capo i dispositivi periferici o unità periferiche, necessari per rendere operativo il sistema.

Control unit, 6, 8, 78
Unità di controllo o di governo; cfr. computer.

Cycle steal, 72
Furto o sottrazione di un ciclo di macchina.

Data
Dati. Talvolta con il significato di operandi o argomenti di operazioni, talvolta con quello pi6 generale di informazioni.

Data structure, 96
Struttura di dati.

Data path, 8, 85
Percorso dei dati. Cfr. *Bus structure.*

Debugging, 106
Individuazione ed eliminazione degli errori contenuti in un programma.

Direct memory access (DMA), 32
Accesso diretto alla memoria; cfr. *I/O operation.*

Disable
Disabilitare. Operazione volta a rendere un circuito o un dispositivo inattivo e insensibile ad eventuali comandi.

Display, 5
Visualizzatore. Dispositivo optoelettronico per la presentazione delle informazioni alla vista di un operatore umano.

Enable
Abilitare. È l'operazione contraria a quella di *Disable.*

Execute phase, 78
Fase d'esecuzione; cfr. *Machine cycle.*

Fetch phase, 78
Fase di prelievo, ossia fase di accesso alla memoria; cfr. *Machine cycle.*

Field
Campo. Parte dell'istruzione dedicata ad un'informazione specifica.

Field-programming, 64
Programmabile dall'utente; cfr. *memory, semiconductor.*

Firmware, 87
Software contenuto nella memoria fissa (ROM) di un'unità centrale (CPU) microprogrammata.

Flag, 14, 52
Indicatore. Registro o cella di memoria per un bit che segnala qualche avvenimento.

Flow-chart, 19
Diagramma di flusso delle operazioni. Ogni operazione stabilita dal programmatore viene rappresentata con un blocco; la successione delle operazioni è indicata dall'ordine con cui sono connessi i blocchi.

Format, 72, 76
Formato. Forma in cui sono presentate le informazioni.

Handshake operation, 29
Letteralmente, funzionamento a stretta di mano. Procedura di colloquio fra due unità non sincronizzate, tipicamente il computer e uni'unità periferica, che si sacmbiano informazioni attraverso un buffer bidirezionale.

Hardware
Insieme delle parti materiali che costituiscono il computer.

Inhibit
Inibire. Lo stesso che *Disable.*

Input/output operations
Operazioni di ingresso/uscita. Riguardano lo scambio di informazioni fra il computer e il mondo esterno. Possono avvenire con differenti modalità:

— a controllo di programma, 30; il computer effettua l'interrogazione ciclica delle unità periferiche allo scopo di riconoscere un'eventuale richiesta di servizio (*polling*);

— ad interruzione del programma (*interrupt*), 31; un dispositivo periferico richiede l'interruzione del programma per essere servito; l'unità di controllo termina l'istruzione in corso, sospende l'esecuzione del programma e si dichiara disponibile all'operazione I/O. L'interruzione è a vettore (*vectored interrupt*, 31); quando è previsto un gruppo di sottoprogrammi di gestione dell'interruzione, ai quali i dispositivi periferici sono variamente associati; è multipla (*multiple interrupt*, 31), se è prevista una gerarchia di livelli di priorità, variamente associati ai dispositivi periferici, che stabilisce l'ordine con cui vengono servite le richieste di interruzione;

— accesso diretto alla memoria (*direct memory access*, DMA, 32); l'unità periferica accede alla memoria senza il tramite della CPU e quindi in un tempo minore; un apposito controller, che in definitiva è un elaboratore più elementare, provvede a sospendere il funzionamento della CPU e a gestire l'operazione I/O.

Instruction code, 13
Istruzione in linguaggio di macchina. Parola binaria, ossia stringa di 0 e 1, che indica le operazioni da eseguire e la posizione fisica dei dati su cui tali operazioni vanno eseguite. Il gruppo di cifre binarie che compongono la parola di istruzione è quindi distinguibile in due sottogruppi: quello che specifica l'operazione da svolgere (*operating code*, *operator*; codice operativo o d'operazione, operatore) e quello che indica la localizzazione fisica degli oggetti dell'operazione (*operandi*; operando, o campo operandi). Le dimensioni e il formato dell'istruzione possono variare (72, 76).

Instruction decoder, 87
Decodificatore dell'istruzione. Interpreta l'istruzione ricopiata dalla memoria nel registro delle istruzioni, informando l'unità di controllo sulla natura delle operazioni successive che questa deve ordinare.

Instruction register, 70, 79
Registro delle istruzioni. Contiene l'istruzione in corso d'esecuzione, che dalla memoria viene ricopiata in questo registro prima di essere eseguita.

Instruction set, 13
Repertorio delle istruzioni. Insieme delle operazioni elementari che un computer specifico è in grado di effettuare. Comprende operazioni di caricamento dell'accumulatore (*loading*, 13), di caricamento in memoria centrale (*storing*, 14), operazioni aritmetiche (13), operazioni logiche (13), operazioni di salto (*jumping*, 14) ossia di abbandono dell'ordine sequenziale nell'esecuzione del programma, ed infine operazioni di ingresso e di uscita (14, 23).

Interrupt, 31
Interruzione. Cfr. I/O operations.

Label, 94
Etichetta. Gruppo di caratteri utilizzato per identificare in modo simbolico una locazione di memoria; consente di ignorare l'indirizzo assoluto di quella locazione.

Language
Linguaggio. Insieme di segni o caratteri e di regole per associarli in parole e in frasi.

machine l., 13; linguaggio di macchina. Linguaggio i cui segni sono 0 e 1; le regole associative dei simboli sono caratteristiche della macchina.

assembly l., symbolic l., 15. Linguaggio assemblativo. Sostituisce ad ogni istruzione del linguaggio di macchina brevi e semplici gruppi di caratteri alfabetici che richiamano la traduzione dell'istruzione in qualche linguaggio naturale. Ha lo scopo di semplificare il lavoro del programmatore, che opera con brevi parole assonanti a parole inglesi invece che con sequenze di 0 e di 1. È ancora un linguaggio proprio della macchina, della quale riflette la struttura; in altri

termini, ogni computer ha un proprio linguaggio assemblativo.

high level l., 17; linguaggio ad alto livello, o linguaggio compilativo. In un tale linguaggio le frasi (*statement*) non si richiamano in nessuna maniera alle operazioni elementari di qualche computer, ma piuttosto alle operazioni che intervengono in determinate categorie di applicazioni. Ha quindi carattere di linguaggio universale, che prescinde da ogni particolare computer; di conseguenza, ogni computer avrà un proprio programma traduttore (*compiler*; compilatore) per la generazione del corrispondente programma in linguaggio di macchina.

Look-up table, 59, 63
Tavola da consultazione.

Loop, 20
Ciclo. Allude ad un segmento di programma ripetuto ciclicamente.

Machine cycle, 78
Ciclo di macchina. L'attività di un computer è organizzata in maniera ciclica attraverso successivi accessi alla memoria centrale e azionamenti dei circuiti che eseguono le operazioni. L'attività unitaria ripetuta prende il nome di ciclo di macchina, ed è appunto distinta in una fase di accesso alla memoria (*fetch phase*: fase di prelievo) e in una fase di azionamento dei circuiti (*execute phase*: fase di esecuzione). L'attività della macchina è scandita dal clock; in ogni periodo di clock la macchina è suscetibile di assumere una configurazione differente dei suoi circuiti, per cui tale periodo definisce uno *stato temporale* (*time state*). La fase di accesso e la fase di esecuzione durano entrambe un multiplo intero di periodi del clock; la durata della fase di prelievo è costante, mentre quella della fase di esecuzione può variare.

Macroinstruction, 16
Macroistruzione. Istruzione in linguaggio assemblativo corrispondente ad un gruppo di istruzioni in linguaggio di macchina.

Memory address register, 70, 78, 85
Registro degli indirizzi di memoria. Contiene l'indirizzo della locazione di memoria alla quale si deve accedere.

Memory data register, 70, 78
Registro dei dati di memoria. Registro buffer nel quale vengono memorizzate temporaneamente le informazioni in entrata e in uscita della memoria centrale.

Memory location, 10
Locazione o posizione in memoria.

Memory, main, 9, 63
Memoria centrale. Sottosistema del computer destinato a contenere il programma.

Memory, semiconductor
Memoria a semiconduttore. È costituita da una matrice di celle ciascuna capace di memorizzare un bit. Alcune sono a lettura e scrittura (RAM, *random access memory*, memoria ad accesso diretto, ossia non sequenziale o non ordinato, 10, 6, 68-69), altre a sola lettura (ROM, *read-only memory*; 10, 63, 68-69); di queste, alcune sono programabili esclusivamente in fabbrica dal costruttore, altre invece possono essere programmate per via elettrica dall' utente (PROM, *programmable read-only memory*, ROM programmabile; 63), o perfino programmate, cancellate e riprogrammate da questo (EPROM, *erasable programmable read-only memory*, ROM programmabile cancellabile, ROM riprogrammabile).

Microcomputer, 88
Computer di dimensioni estremamente ridotte costruito intorno ad un microprocessore.

Microprocessor, 10, 11
Microprocessore o microelaboratore. Modulo (o gruppo di moduli) integrato contenente unità aritmetico-idrica e unità di controllo.

Microprogramming, 87
Microprogrammazione. Modo di organizzare l'esecuzione delle operazioni elementari del repertorio di un computer (operazioni di macchina). Mentre in un computer non microprogrammato queste operazioni sono effettuate da una

circuiteria fissa dedicata a ciascuna di esse, in un computer microprogrammato queste sono risolte in operazioni sub-elementari, effettuate da una circuiteria programmabile. La microprogrammazione comporta quindi la presenza in seno alla CPU di un ulteriore elaboratore più elementare, dotato di una memoria fissa programmata (ROM); il software contenuto entro questa memoria prende il nome di *firmware*.

Minicomputer, 2
Computer di dimensioni e di potenza di calcol ridotte; può essere costruito intorno ad un microprocessore. Si colloca in posizione intermedia fra un computer e un microcomputer.

Multiplexer, bidirectional, 5, 24
Multiplexer bidirezionale. Dispositivo che consente la concentrazione e la diramazione delle vie su cui transitano le informazioni.

Multiplexer, unidirectional, 5, 24
Multiplexer unidirezionale. Consente solo l'una o l'altra delle operazioni del multiplexer bidirezionale.

Multiprocessing, 106
Multielaborazione. Attività di un computer con più di un'unità centrale.

Multiprogramming, 106
Multiprogrammazione. Esecuzione da parte di una sola unità centrale di più programmi, contenuti nella stessa memoria centrale. L'esecuzione dei programmi apparentemente è simultanea, ma effettivamente è tale solo nei momenti in cui i programmi diversi richiedono unità diverse; altrimenti, l'unità centrale intercala attività relative ai differenti programmi.

Operand, 13
Operando; cfr. *Instruction code.*

Operating code, operation code, 13
Codice d'operazione o codice operativo; cfr. *Instruction code.*

Operator, 13
Operatore; cfr. *Instruction code.*

Overflow, 52, 98, 100
Superamento di capacità.

Overhead
Penalizzazione.

Parity code, even, 46
Codice a parità.

Parity code, odd, 46
Codice a disparità.

Pointer, 97, 98, 99
Puntatore.

Polling, 30
Interrogazione; cfr. *I/O operations.*

Program, 12
Programma. Lista di istruzioni o di frasi, eseguibili e non eseguibili, espresse in qualche linguaggio ammesso dalla macchina.

Program, object, 107
Programma oggetto. Programma in linguaggio di macchina (ossia in un codice binario) ottenuto dalla traduzione del programma sorgente; la traduzione viene effetuata automaticamente dallo stesso o da un altro computer guidato da un apposito programma traduttore.

Program, source, 94
Programma sorgente. Programma scritto in un linguaggio diverso dal linguaggio di macchina; deve essere tradotto in un programma oggetto.

Program counter, 8, 70
Contatore di programma o contatore delle istruzioni. Registro nel quale è contenuto l'indirizzo della locazione di memoria alla quale accedere nella successiva fase di fetch. Il contenuto di questo contatore progredisce di un'unità ad ogni ciclo di macchina, salvo nel caso di un jump.

Programmable logic array (PLA), 88
Matrice logica programmabile.

Pseudoinstruction, 92
Pseudoistruzione. Istruzione non eseguibile, che non corrisponde cioè ad operazioni da fare.

Queue, 98
Coda.

Refreshing, 66, 70
Rinfrescamento.

Register
Registro. Memoria di dimensioni estremamente ridotte, destinato a contenere una sola parola di uno o più bit, con significato specializzato.

Round-off, 37
Arrotondamento.

Scrotch-pad memory, 6
Memoria temporanea, utilizzata per memorizzare provvisoriamente risultati parziali o intermedi che si producono nel corso delle operazioni.

Software
L'insieme dei programmi memorizzati entro un computer. Viene distinto in *operating software* (software operativo), che comprende l'insieme dei programmi di servizio, e in *application software* (software applicativo), che è il programma d'utente. Il software operativo può comprendere un programma *loader*, che provvede al caricamento dei programmi d'utente, un programma *debugger*, che segnale eventuali errori formali presenti nel programma, un programma *editor*, che assiste nella fase iniziale di messa a punto del programma consentendo cambiamenti del testo.

Stack, 96
Pila.

Statement, 17
Frase in linguaggio compilativo o ad alto livello.

Statement, non executable
Statement non eseguibile. Frase dichiarativa alla quale non corrispondono operazioni da compiere.

Subroutine, 18
Sottoprogramma.

Substractor, binary, 54
Sottrattore binario.

sign-and-magnitude s., 54: s. in segno e modulo,

full s., 54; s. completo,

look-ahead borrow s., 55; s. a prestito simultaneo,

1's complement s., 55; s. in completo ad uno,

2's complement s., 56; s. in complemento a due.

Tristate, 25, 68
Moduli integrati digitali che in condizione di attività presentano in uscita bassa impedenza, con tensione alta o con tensione bassa, come i circuiti digitali convenzionali, ma che inoltre possiedono una condizione di inattività, nella quale presentano elevata impedenza di uscita. Sono componenti essenziali della struttura a bus, perché possono venire connessi al bus o disconnessi da questo dietro comando elettrico.

Underflow, 98, 100
Sottosvuotamento.

Word, 10
Parola. Insieme di bit trattati simultaneamente dalla macchina, e per questo considerati come un'unità. Il numero dei bit viene detto la lunghezza della parola (*ward length*, 6). Le parole di otto bit vengono dette anche *byte*.

Working register, 6
Registro operativo. I registri operativi entrano nel gruppo di registri contenuti entro la CPU con funzione di scratchpad memory.

C.E.L.I.

MICROCOMPUTER E MICROPROCESSORI

A. BARNA - D. I. PORAT